How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction

YUMING ZHOU, YIBIAO YANG, HONGMIN LU, LIN CHEN, YANHUI LI, and YANGYANG ZHAO, Nanjing University
JUNYAN QIAN, Guilin University of Electronic Technology
BAOWEN XU, Nanjing University

Background. Recent years have seen an increasing interest in cross-project defect prediction (CPDP), which aims to apply defect prediction models built on source projects to a target project. Currently, a variety of (complex) CPDP models have been proposed with a promising prediction performance.
Problem. Most, if not all, of the existing CPDP models are not compared against those simple module size models that are easy to implement and have shown a good performance in defect prediction in the literature.
Objective. We aim to investigate how far we have really progressed in the journey by comparing the performance in defect prediction between the existing CPDP models and simple module size models.
Method. We first use module size in the target project to build two simple defect prediction models, ManualDown and ManualUp, which do not require any training data from source projects. ManualDown considers a larger module as more defect-prone, while ManualUp considers a smaller module as more defect-prone. Then, we take the following measures to ensure a fair comparison on the performance in defect prediction between the existing CPDP models and the simple module size models: using the same publicly available data sets, using the same performance indicators, and using the prediction performance reported in the original cross-project defect prediction studies.
Result. The simple module size models have a prediction performance comparable or even superior to most of the existing CPDP models in the literature, including many newly proposed models.
Conclusion. The results caution us that, if the prediction performance is the goal, the real progress in CPDP is not being achieved as it might have been envisaged. We hence recommend that future studies should include ManualDown/ManualUp as the baseline models for comparison when developing new CPDP models to predict defects in a complete target project.

CCS Concepts: • **Software and its engineering → Software evolution**; **Maintaining software**;

Additional Key Words and Phrases: Defect prediction, cross-project, supervised, unsupervised, model

This work is partially supported by the National Key Basic Research and Development Program of China (2014CB340702) and the National Natural Science Foundation of China (61432001, 61772259, 61472175, 61472178, 61702256, 61562015, 61403187).

Authors' addresses: Y. Zhou (corresponding author), Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, and B. Xu (corresponding author), State Key Laboratory for Novel Software Technology, Nanjing University, No. 163, Xianlin Road, Nanjing, 210023, Jiangsu Province, P.R. China; emails: {zhouyuming, yangyibiao, hmlu, lchen, yanhuili, bwxu}@nju.edu.cn, csurjzhyy@163.com; Y. Qian, Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, 541004, Guangxi Province, P.R. China; email: qjy2000@gmail.com.

https://doi.org/10.1145/3183339

ACM Reference format:
Yuming Zhou, Yibiao Yang, Hongmin Lu, Lin Chen, Yanhui Li, Yangyang Zhao, Junyan Qian, and Baowen Xu. 2018. How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction. *ACM Trans. Softw. Eng. Methodol.* 27, 1, Article 1 (April 2018), 51 pages.
https://doi.org/10.1145/3183339

1 INTRODUCTION

A defect prediction model can predict the defect-proneness of the modules (e.g., files, classes, or functions) in a software project. Given the prediction result, a project manager can (1) classify the modules into two categories, high defect-prone or low defect-prone, or (2) rank the modules from the highest to lowest defect-proneness. In both scenarios, the project manager could allocate more resources to inspect or test those high defect-prone modules. This will help to find more defects in the project if the model can accurately predict defect-proneness. For a given project, it is common to use the historical project data (e.g., module-level complexity metric data and defect data in the previous releases of the project) to train the model. Prior studies have shown that the model predicts defects well in the test data if it is trained using a sufficiently large amount of data [23]. However, in practice, it might be difficult to obtain sufficient training data [129]. This is especially true for a new type of projects or projects with little historical data collected.

One way to deal with the shortage of training data is to leverage the data from other projects (i.e., source projects) to build the model and apply it to predict the defects in the current project (i.e., the target project) [9]. However, in practice, it is challenging to achieve accurate cross-project defect prediction (CPDP) [129]. The main reason is that the source and target project data usually exhibit significantly different distributions, which violates the similar distribution assumption of the training and test data required by most modeling techniques [61, 80, 123]. Furthermore, in many cases, the source and target project data even consist of different metric sets, which makes it difficult to use the regular modeling techniques to build and apply the prediction model [17, 27, 41, 79]. In recent years, various techniques have been proposed to address these challenges and a large number of CPDP models have hence been developed (see Section 2.4). In particular, it has been reported that these CPDP models produce a promising prediction performance [27, 41, 61, 79, 80].

Yet, most, if not all, of the existing CPDP models are not compared against those simple module size models that are easy to implement and have shown a good performance in defect prediction in the literature. In the past decades, many studies have reported that simple models based on the module size (e.g., SLOC, source lines of code) in a project can in general predict the defects in the project well [2, 49–51, 65, 102, 127, 128]. In the context of CPDP, we can use module size in a target project to build two simple module size models, ManualDown and ManualUp [49–51, 65]. The former considers a larger module as more defect-prone, while the latter considers a smaller module as more defect-prone. Since ManualDown and ManualUp do not require any data from the source projects to build the models, they are free of the challenges on different data distributions/metric sets of the source and target project data. In particular, they have a low computation cost and are easy to implement. In contrast, due to the use of complex modeling techniques, many existing CPDP models not only have a high computation cost but also involve a large number of parameters needed to be carefully tuned. This imposes substantial barriers to apply them in practice, especially for large projects. Furthermore, previous studies show that module size has a strong confounding effect on the associations between code metrics and defect-proneness [20, 128]. Emam et al. even reported that their associations disappeared after controlling for module

size [20]. This indicates that module size may have a capability in defect prediction similar to code metrics, the most commonly used predictors in the existing CPDP models. An interesting question hence arises: *"If the prediction performance is the goal, then how do the existing CPDP models perform compared with ManualDown and ManualUp?"* This question is important for both practitioners and researchers. For practitioners, it will help to determine whether it is worth to apply the existing CPDP models in practice. If simple module size models perform similarly or even better, then there is no practical reason to apply complex CPDP models. For researchers, if simple module size models perform similarly or even better, then there is a strong need to improve the prediction power of the existing CPDP models. Otherwise, the motivation of applying those CPDP models could not be well justified. To the best of our knowledge, of the existing CPDP studies, only two studies compared the proposed CPDP models against simple module size models [9, 12]. In other words, for most of the existing CPDP models, it remains unclear whether they have a superior prediction performance compared with simple module size models.

In this article, we attempt to investigate how far we have really progressed in the journey by conducting a comparison in defect prediction performance between the existing CPDP models and simple module size models. We want to know not only whether the difference in defect prediction performance is statistically significant but also whether it is of practical importance. In our study, we take the following two measures to ensure a fair comparison. On the one hand, we use the same publicly available datasets and the same performance indicators to collect the performance values for the simple module size models as those used in the existing CPDP studies. On the other hand, we do not attempt to re-implement the CPDP models to collect their prediction performance values. In contrast, we use the prediction performance values reported in the original CPDP studies to conduct the comparison. Therefore, the implementation bias can be avoided. These measures ensure that we can draw a reliable conclusion on the benefits of the existing CPDP models w.r.t. simple module size models in defect prediction performance in practice.

Under the above experimental settings, we perform an extensive comparison between the existing CPDP models and simple module size models for defect prediction. Surprisingly, our experimental results show that simple module size models have a prediction performance comparable or even superior to most of the existing CPDP models in the literature, including many newly proposed models. Consequently, for practitioners, it would be better to apply simple module size models rather than the existing CPDP models to predict defects in a target project. This is especially true when taking into account the application cost (including metrics collection cost and model building cost). The results reveal that, if the prediction performance is the goal, the current progress in kjjc zCPDP studies is not being achieved as it might have been envisaged. We hence strongly recommend that future CPDP studies should consider simple module size models as the baseline models to be compared against. The benefits of using a baseline model are two-fold [112]. On the one hand, this would ensure that the predictive power of a newly proposed CPDP model could be adequately compared and assessed. On the other hand, "the ongoing use of a baseline model in the literature would give a single point of comparison". This will allow a meaningful assessment of any new CPDP model against previous CPDP models.

The rest of this article is organized as follows. Section 2 introduces the background on cross-project defect prediction, including the problem studied, the general framework, the performance evaluation indicators, and the state of progress. Section 3 describes the experimental design, including the simple module size model, the research questions, the datasets, and the data analysis method. Section 4 presents in detail the experimental results. Section 5 compares the simple module size models with related work. Section 6 discusses the results and implications. Section 7 analyzes the threats to the validity of our study. Section 8 concludes the article and outlines directions for future work.

2 CROSS-PROJECT DEFECT PREDICTION

In this section, we first describe the problem that cross-project defect prediction aims to address. Then, we present a general framework that depicts the key components in a supervised cross-project defect prediction. After that, we introduce the commonly used performance evaluation indicators involved in the existing CPDP studies. Finally, we give a literature overview of current developments in this area and provide a comparison of the main CPDP studies under the general framework.

2.1 Problem Statement

The purpose of cross-project defect prediction is to address the shortage problem of training data that are used for building a model to predict defects in a target release of a target project. To predict defects in the target release, it is common to use a two-phase process (i.e., model building and application). At the model-building phase, the metric data and the defect data are first collected from the modules in historical releases (i.e., the releases before the target release in the target project). Then, based on the collected data, a specific modeling technique is used to train a model to capture the relationships between the metrics and defect-proneness. At the model application phase, the same metrics are first collected from the modules in the target release. Then, the predicted defect-proneness of each module in the target release is obtained by substituting the corresponding metric data into the model. After that, the predicted performance is evaluated by comparing the predicted defect-proneness against the actual defect information in the target release. It has been shown that, if there is a sufficiently large amount of training data at the model-building phase, the resulting model in general predicts defects well at the model application phase [23]. In practice, however, sufficiently large training data are often unavailable, especially for those projects with few or even no historical releases. To mitigate this problem, researchers propose to use the training data from other projects (i.e., source projects) to build the model for the target project, which is called cross-project defect prediction [9, 106, 129].

In the literature, it has been highlighted that there are two major challenges that cross-project defect prediction has to deal with. The first challenge is that the source and target project data usually exhibit significantly different distributions [13, 41, 42, 61, 80, 123]. Since the source and target projects might be from different corporations, there might have a large difference between their development environments. Consequently, the same metrics in the source and target project data might have a large difference between the data distributions. However, regular modeling techniques are based on the assumption that the training and test data are drawn from the same distribution. Therefore, it is difficult to achieve a good prediction performance by directly applying regular modeling techniques to build the prediction models. The second challenge is that the source and target project data might have no common metrics [27, 41, 79]. In practice, since the source and target projects might be from different corporations, it is likely that the source and target project data consist of completely different metric sets. However, regular modeling techniques are based on the assumption that the training and test data have common metric sets. Consequently, it is difficult, if not impossible, to predict defects in the target project using the models built with regular modeling techniques on the source project data. In recent years, much effort has been devoted to dealing with these challenges.

2.2 General Framework

In the field of cross-project defect prediction, supervised models are the mainstream models. Figure 1 presents a general framework that applies supervised techniques to cross-project defect prediction. As can be seen, the target project consists of k release, among which the kth release is

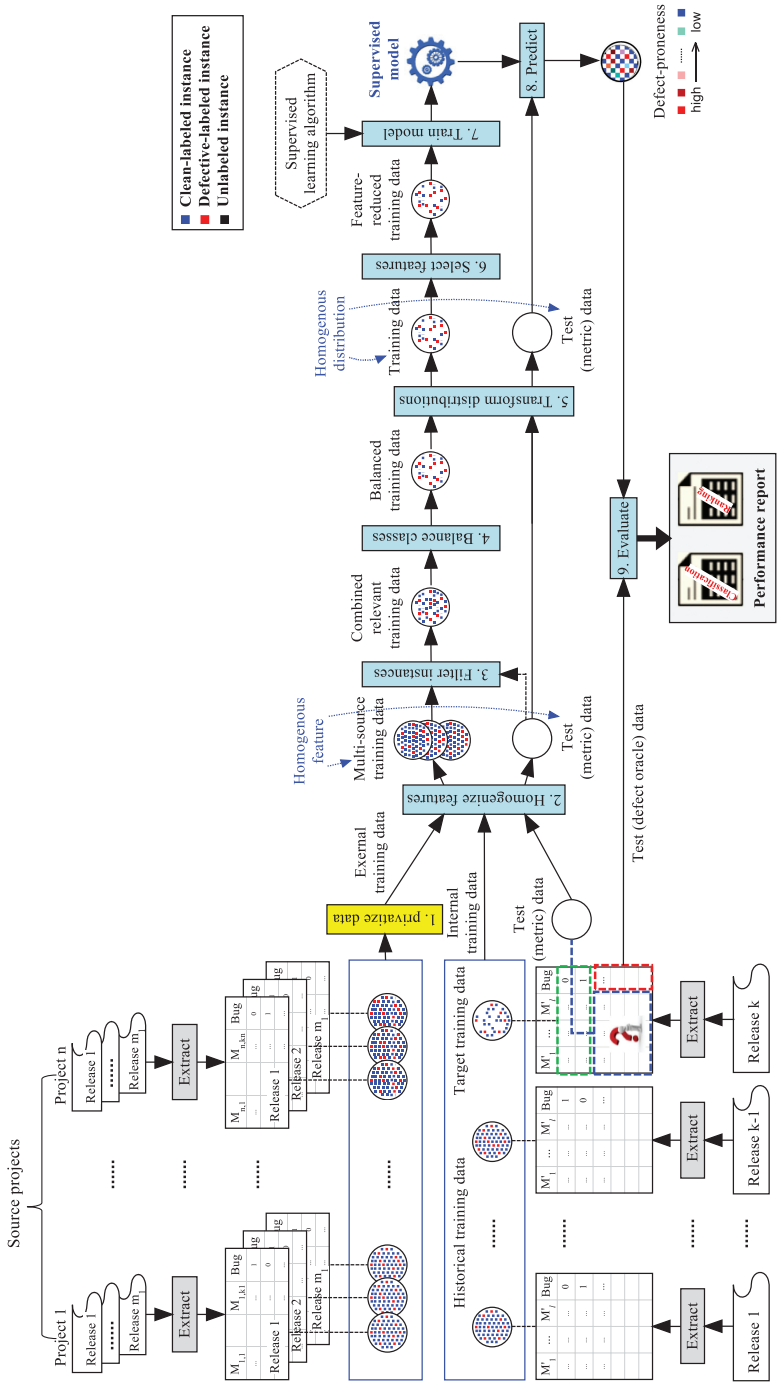

Fig. 1. The general framework that applies supervised modeling techniques to cross-project defect prediction.

the target release and the 1th, . . . , $k-1$th releases are the historical releases. The source projects consist of n projects, each having a number of releases. Each release in the target and source projects has an associated dataset, which consists of the metric data and the defect label data at the module level. The modules can be packages, files, classes, or functions, depending on the defect prediction context. The metric sets are assumed to be the same for all releases within a single project but can be different for different projects. In each dataset, one instance represents a module and one feature represents a software metric extracted from the module.

From Figure 1, on the one hand, we can see that the *training data* consist of the external training data and the internal training data. The external training data are indeed the labeled source project data. The internal data consist of two parts: the *historical training data* (i.e., the labeled historical release data) and the *target training data* (i.e., a small amount of the labeled target release data of the target project). On the other hand, we can see that the *test data* consist of the labeled target release data excluding the target training data. In other words, the labeled target release data are divided into two parts: a small amount of data used as the target training data and the remaining data used as the test data. For the simplicity of presentation, we use the *test metric data* and the *defect oracle data* to respectively denote the metric data and the defect label data in the test data. The test metric data will be introduced into the model built with the training data to compute the predicted defect-proneness. Once the predicted defect-proneness of each module in the test data is obtained, it will be compared against the defect oracle data to compute the prediction performance.

At a high level, the application of a supervised CPDP model to a target project in practice involves the following three phases: data preparation, model training, and model testing. At the data preparation phase, preprocess the training data collected from different sources to make them appropriate for building a CPDP model for the target project. On the one hand, there is a need to address the privacy concerns of (source project) data owners, i.e., preventing the disclosure of specific sensitive metric values of the original source project data [84, 85]. On the other hand, there is a need to address the utility of the privatized source project data in cross-project defect prediction. This includes dealing with homogenous feature sets (i.e., metric set) between source and target project data [27, 41, 79], filtering out irrelevant training data [39, 86, 94, 106], handling class-imbalanced training data [42, 93], making source and target project have a similar data distribution [13, 41, 42, 61, 80, 123], and removing irrelevant/redundant features [5, 9, 26, 44]. At the modeling training phase, use a supervised modeling technique to build a CPDP model [59, 96, 108, 116, 126]. In other words, this aims to use a specific supervised learning algorithm to build a model to capture the relationship between the metrics (i.e., the independent variables) and defect-proneness (i.e., the dependent variable) in the training data. At the model testing phase, apply the CPDP model to predict defects in the target release and evaluate its prediction performance under the classification or ranking scenario. In the former scenario, the modules in the test data are classified into defective or not defective. In the latter scenario, the modules in the test data are ranked from the highest to the lowest predicted defect-proneness. Under each scenario, the performance report is generated by comparing the predicted defect-proneness with the defect oracle data. By the above phases, it is expected that the knowledge about the effect of the metrics on defects can be learned from the source projects by the supervised model to predict defects in the target release of the target project.

2.3 Performance Evaluation

Table 1 summarizes the prediction performance indicators involved in the existing cross-project defect prediction literature. The first column reports the scenario in which a specific indicator is used. The second, third, and fourth columns, respectively, show the name, the definition, and the

Table 1. The Prediction Performance Indicators Involved in the Existing Cross-Project Defect Prediction Studies

Scenario	Name	Definition	Interpretation	Better
Classification	Recall	TP / (TP + FN)	The fraction of defective modules that are predicted as defective	High
	Precision	TP / (TP + FP)	The fraction of predicted defective modules that are defective	High
	PD	TP / (TP + FN)	Probability of Detection. Same as Recall	High
	PF	FP / (FP + TN)	Probability of False alarm. The fraction of not defective modules that are predicted as defective	Low
	Correctness	TP / (TP + FP)[9]	Same as Precision	High
	Completeness	Defects(TP) / Defects(TP + FN)[9]	A variant of Recall. The percentage of defects found	High
	F_β	$(1+\beta^2)\times$Recall$\times$Precision/ (Recall + $\beta^2\times$Precision)	The harmonic mean of precision and recall ($\beta = 1$ or 2)	High
	G_1	[2×PD×(1-PF)]/[PD+(1-PF)]	The harmonic mean of PD and 1-PF	High
	G_2	$\sqrt{Recall \times Precision}$	The geometric mean of recall and precision	High
	G_3	$\sqrt{Recall \times (1-PF)}$	The geometric mean of recall and 1-PF	High
	Balance	$1-\frac{\sqrt{(0-PF)^2+(1-PD)^2}}{\sqrt{2}}$ [107]	The balance between PF and PD	High
	ED	$\sqrt{\theta \times (1-PD)^2 + (1-\theta)\times PF^2}$ [52]	The distance between (PD, PF) and the ideal point on the ROC space (1, 0), weighted by cost function θ ($\theta = 0.6$ in default)	Low
	MCC	$\frac{TP\times TN - FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$ [123]	A correlation coefficient between the actual and predicted binary classifications	High
	AUC	The area under ROC curve [24]	The probability that a model will rank a randomly chosen defective module higher than a randomly chosen not defective one	High
	NECM	$(FP + C_{\text{II}}/C_{\text{I}\times FN})/(TP + FP + TN + FN)$[48]	The normalized expected cost of misclassification	Low
	Z*	$\frac{(o-e)\sqrt{n}}{e(n-e)}$ [10]	A statistic for assessing the statistical significance of the overall classification accuracy. Here, o is the total number of correct classifications (i.e., o = TP + TN), e is the expected number of correct classification due to chance, and n is the total number of instances (i.e., n = TP + FP + TN + FN)	High
Ranking	AUCEC	Area(m): the area under the cost-effectiveness curve corresponding to the model m [90]	The cost-effectiveness of the overall ranking	High
	NofB20	The number of defective modules in the top 20% SLOC	The cost-effectiveness of the top ranking	High
	PofB20	The percentage of defects in the top 20% SLOC	The cost-effectiveness of the top ranking	High
	FPA	The average, over all values of k, of the percentage of defects contained in the top k modules [111]	The effectiveness of the overall ranking	High

(Continued)

Table 1. Continued

Scenario	Name	Definition	Interpretation	Better
	E1(R)	The percentage of modules having R% defects	The effectiveness of the top ranking	Low
	E2(R)	The percentage of code having R% defects	The cost-effectiveness of the top ranking	Low
	Prec@k	Top k precision	The fraction of the top k ranked modules that are defective	High

informal interpretation for each specific indicator. The last column indicates the expected direction for a better prediction performance. As can be seen, many performance indicators have a similar or even identical meaning. For example, PD has the same meaning as Recall, while Completeness is a variant of Recall. In the next section, we will compare the existing CPDP studies, including the performance indicators used. For the performance indicators, we use the exact same names as used in the original studies. This will help understand the comparison, especially for those readers who are familiar with the existing CPDP studies. Therefore, we include these indicators in Table 1, even if they express a similar or even identical meaning.

Classification performance indicators. When applying a prediction model to classify modules in practice, we first need to determine a classification threshold for the model. A module will be classified as defective if the predicted defect-proneness is larger than the threshold and otherwise it will be classified as not defective. The four outcomes of the classification using the threshold are as follows: TP (the number of modules correctly classified as defective), TN (the number of modules correctly classified as not defective), FP (the number of modules incorrectly classified as defective), and FN (the number of modules incorrectly classified as not defective). As shown in Table 1, all classification performance indicators except AUC (Area Under ROC Curve) are based on TP, TN, FP, and FN, i.e., they depend on the threshold. AUC is a threshold-independent indicator, which denotes that the area under a ROC (Relative Operating Characteristic) curve. A ROC curve is a graphical plot of the PD (the y-axis) vs. PF (the x-axis) for a binary classification model as its decision threshold is varied. In particular, all classification performance indicators except NECM do not explicitly take into account the cost of misclassification caused by the prediction model.

Ranking performance indicators. In the ranking scenario, the performance indicators can be classified into two categories: effort-aware and non-effort-aware. The former includes AUCEC, NofB20, PofB20, and E2(R), while the latter includes FPA, E1(R), and Prec@k. The effort-aware indicators take into account the effort required to inspect or test those modules predicted as defective to find whether they contain defects. In these indicators, the size of a module measured by SLOC (source lines of code) is used as the proxy of the effort required to inspect or test the module. In particular, AUCEC is based on the SLOC-based Alberg diagram [6]. In such an Alberg diagram, each defect-proneness prediction model corresponds to a curve constructed as follows. First, the modules in the target release are sorted in decreasing order according to the predicted defect-proneness by the model. Then, the cumulative percentage of SLOC of the top modules selected from the module ranking (the x-axis) is plotted against the cumulative percentage of defects found in the selected top modules (the y-axis). AUCEC is the area under the curve corresponding to the model. Note that NofB20, PofB20, and E2(R) quantify the performance of the top ranking, while AUCEC quantifies the performance of the overall ranking. The non-effort-aware performance indicators do not take into account the inspection or testing effort. Of non-effort-aware indicators, E1(R) and Prec@k quantify the performance of the top ranking, while FPA quantifies the performance of the overall ranking.

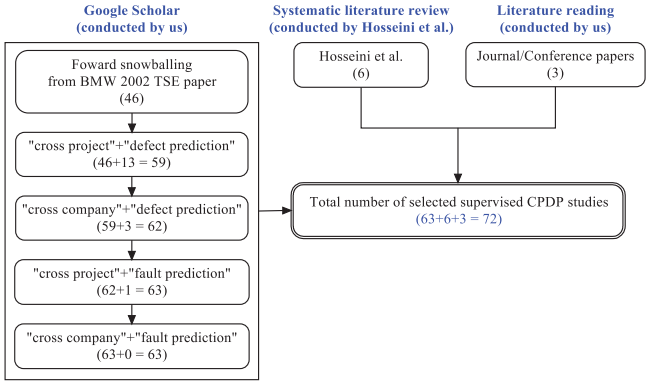

Fig. 2. The process to select supervised CPDP studies.

2.4 State of Progress

To understand the progress in supervised cross-project defect prediction, we conducted a search of the literature published between 2002 and 2017.[1] The starting year of the search was set to 2002 as it is the year that the first CPDP article was published [9] (abbreviated as "BMW 2002 TSE paper" in the following). Before the search, we set up the following inclusion criteria: (1) the study was a supervised CPDP study; (2) the study was written in English; (3) the full text was available; (4) only the journal version was included if the study had both the conference and journal versions; and (5) the prediction scenario was classification or ranking.

As shown in Figure 2, we searched the articles from three sources: Google Scholar, the existing systematic literature reviews, and literature reading. We used Google Scholar as the main search source, as it "provides a simple way to broadly search for scholarly literature."[2] First, we did a forward snowballing search [114] by recursively examining the citations of the "BMW 2002 TSE paper". Consequently, 46 relevant articles were identified [1, 3, 5, 7, 9, 10, 12, 13, 15, 17, 26–30, 37, 39, 41, 42, 44, 46, 47, 52, 59, 66, 79, 80, 82, 85–87, 89, 90, 92–96, 107, 109, 116, 118, 120, 122, 123, 126]. Then, we used "cross project" + "defect prediction" as the search terms to do a search. As a result, 13 additional relevant articles [14, 31, 32, 43, 45, 61, 71, 84, 88, 99, 101, 108, 129] were found with respect to the identified 46 articles. Next, we used "cross company" + "defect prediction" as the search terms, identifying 3 additional relevant articles [104, 106, 110] with respect to the 59 (=46 + 13) articles. After that, we used "cross project" + "fault prediction" as the search terms, identifying 1 additional relevant article [100] with respect to the 62 (=59 + 3) articles. Finally, we used the terms "cross company" + "fault prediction" and did not find any additional relevant article. By the above steps, we totally identified 63 (=46 + 13 + 3 + 1 + 0) supervised CPDP articles from Google Scholar. In addition to Google Scholar, we identified 6 additional relevant articles from a systematic literature review by Hosseini et al. [40]. Hosseini et al. identified 46 primary CPDP studies from three electronic databases (the ACM digital library, the IEEE Explore, and the ISI Web of Science) and two search engines (Google Scholar and Scopus). After applying our inclusion criteria to the 46 CPDP studies, we found that 6 relevant articles [60, 73, 74, 103, 105, 119] were not present in the 63 identified CPDP articles from Google Scholar. The last source was literature reading, through which we found 3 additional relevant articles [48, 72, 121] with respect

[1]The literature search was conducted on April 21, 2017.
[2]https://scholar.google.com/intl/en/scholar/about.html.

to the 69 (=63 + 6) CPDP articles. Therefore, the overall search process resulted in 72 (=63 + 6 +3) supervised CPDP articles found in the literature.

Table 2 provides an overview of the main literature concerning supervised cross-project defect prediction. For each study, the 2nd and 3rd columns, respectively, list the year published and the topic involved. The 4th to 6th columns list the source and target project characteristics, including the number of source/target projects (releases) and the programming languages involved. For the simplicity of presentation, we do not report the number of releases in parentheses if it is equal to the number of projects. Note that the entries with gray background under the 4th and 5th columns indicate that the source and target projects are different projects. The 7th to 12th columns list the key modeling components (challenges) covered. A "Yes" entry indicates that the study explicitly considers the corresponding component and a blank entry indicates that it does not. The 13th to 16th columns list the performance evaluation context, including the use (or not) of the target training data (i.e., a small amount of the labeled target release data used as the training data), the application scenarios investigated, the main performance indicators employed, and the public availability of test data. An entry with gray background under the 15th column indicates that the study only graphically visualizes the performance and does not report numerical results. The 17th column reports the use (or not) of simple module size models that classify or rank modules in the target release by their size as the baseline models. The last column indicates whether the CPDP model proposed in the study will be compared against the simple module size models described in Section 3 in our study.

From Table 2, we have the following observations. First, the earliest CPDP study appears to be Briand et al.'s work [9]. In their work, Briand et al. investigated the applicability of CPDP on object-oriented software projects. Their results showed that a model built on one project produced a poor classification performance but a good ranking performance on another project. This indicates that, if used appropriately, CPDP would be helpful for practitioners. Second, CPDP has attracted a rapid increasing interest over the past few years. After Briand et al.'s pioneering work, more than 70 CPDP studies are published. The overall results show that CPDP has a promising prediction performance, comparable to or even better than WPDP (within-project defect prediction). Third, the existing CPDP studies cover a variety of wide-ranging topics. These topics include validating CPDP on different development environments (open-source and proprietary), different development stages (design and implementation), different programming languages (C, C++, C#, Java, JS, Pascal, Perl, and Ruby), different module granularity (change, function, class, and file), and different defect predictors (semantic, text, and structural). These studies make a significant contribution to our knowledge about the wide application range of CPDP. Fourth, the number of studies shows a highly unbalanced distribution on the key CPDP components. Of the key components, "filter instances" and "transform distributions" are the two most studied components, while "privatize data" and "homogenize features" are the two less studied components. In particular, none of the existing CPDP studies covers all of the key components and most of them only take into account less than three key components. Fifth, most studies evaluate CPDP models on the complete target release data, which are publicly available, under the classification scenario. In few studies, only a part of the target release data are used as the test data. The reason is either that there is a need to use the target training data when building a CPDP model or that a CPDP model is evaluated on the same test data as the WPDP models. Furthermore, most studies explicitly report the prediction results in numerical form. This allows us to make a direct comparison on the prediction performance of simple module size models against most of the existing CPDP models without a re-implementation. Finally, of 72 studies, only two studies (i.e., Briand et al.'s study [9] and Canfora et al.'s study [12]) used simple module size models in the target release as the baseline models. Both studies reported that the proposed

Table 2. Literature Overview of Cross-Project Defect Prediction

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Zhang et al. [122]	2017	Distribution transformation	18	18	Java			Yes		Yes	Yes		Classification	F_1, AUC	All		Yes
Stuckman et al. [101]	2017	Vulnerability prediction	6	3	Php		Yes	Yes	Yes		Yes		Classification	F_1	All		Yes
Aarti et al. [1]	2017	Utility analysis in prediction	17(35)	17(35)	C/C++/Java			Yes			Yes	Yes	Classification	Precision, recall	Partial		Yes
Herbold et al. [37]	2017	Local prediction models	48(79)	48(79)	C/C++/Java			Yes	Yes	Yes			Classification	F_1, AUC	All		Yes
Ryu et al. [95]	2017	Combination of TL and CIL	15	15	Java				Yes	Yes		Yes	Classification	G_3, Balance	Not		No
Jing et al. [42]	2017	Class imbalance learning	16	16	C/C++/Java			Yes	Yes	Yes			Classification	F2, AUC	All		Yes
Krishna et al. [52]	2016	Bellwether effect	23(51)	23(51)	C/C++/Java			Yes	Yes				Classification	ED	All		Yes
Xia et al. [116]	2016	Compositional model	10(29)	10(29)	Java			Yes				Yes	Both	F_1, PofB20	All		Yes
Zhang et al. [123]	2016	Universal model	1385	5	C/Pascal/C++/Java/C#					Yes			Classification	AUC	All		Yes
Kamei et al. [44]	2016	Just-in-time prediction	11	11	Java/JS/C/C++/Perl/Ruby			Yes	Yes	Yes	Yes		Classification	AUC	Partial		No
Wang et al. [109]	2016	Learning semantic features	10(26)	10(11)	Java			Yes	Yes				Classification	F_1	All		Yes
Ryu et al. [93]	2016	Class imbalance learning	10	10	C/C++/Java				Yes	Yes			Classification	AUC	Partial		Yes

(Continued)

Table 2. Continued

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Ryu et al. [92]	2016	Multi-objective optimization	13	13	Java			Yes		Yes			Classification	Balance, AUC	All		Yes
Hosseini et al. [39]	2016	Training data selection	13	13	Java			Yes					Classification	F_1, G_2	All		Yes
Moshtari et al. [71]	2016	Vulnerability predictors	5	5	C/C++/Java								Classification	AUC, F2	Not		No
You et al. [118]	2016	Rank-oriented model	15(39)	15(39)	Java			Yes		Yes			Ranking	Prec@10	All		Yes
Cheng et al. [17]	2016	Heterogeneous prediction	14	14	C/Java		Yes	Yes		Yes			Classification	F_1, AUC	All		Yes
Yu et al. [121]	2016	Transfer defect learning	15	15	Java			Yes		Yes		Yes	Classification	G_1	Not		No
Kaur et al. [46]	2016	Text-based prediction	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Classification	N/A	N/A		No
Kaur et al. [45]	2016	Value of academic projects	17	4(16)	Java			Yes					Classification	AUC	All		Yes
Catal et al. [14]	2016	Threshold-based prediction	6	6	C/C++/Java						Yes		Classification	AUC	All		Yes
Yu et al. [120]	2016	Filter vs. feature selection	6	6	C/Java			Yes			Yes		Classification	AUC	All		No
Porto et al. [87]	2016	Filter with feature selection	11(36)	11(36)	Java			Yes			Yes		Both	G_1, FPA	Not		No
He et al. [29]	2016	New relevancy filter method	15	15	Java			Yes		Yes			Classification	AUC	All		Yes

(Continued)

Table 2. Continued

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Ryu et al. [94]	2015	Hybrid instance selection	7	7	C/C++/Java			Yes	Yes	Yes			Classification	Balance	Partial		Yes
Kawata et al. [47]	2015	New relevancy filter method	12(35)	18(21)	Java			Yes					Classification	F_1, G_1, AUC	All		Yes
He et al. [26]	2015	Simplified metric set	10(34)	10(34)	Java			Yes			Yes		Classification	F_1	All		Yes
Chen et al. [15]	2015	Negative samples reduction	15	15	Java			Yes	Yes	Yes		Yes	Classification	G_1, MCC	Not		No
Canfora et al. [12]	2015	Multi-objective optimization	10	10	Java					Yes			Ranking	AUCEC	All	Yes	Yes
Peters et al. [85]	2015	Privacy-preserving data sharing	7	10	Java	Yes		Yes		Yes			Classification	G_1	All		Yes
Jing et al. [41]	2015	Heterogeneous prediction	14	14	C/Java		Yes	Yes		Yes			Classification	F_1	All		Yes
Nam et al. [79]	2015	Heterogeneous prediction	28	28	C/Java		Yes				Yes		Classification	AUC	All		Yes
Qin et al. [89]	2015	Aging related defect prediction	7	7	C/C++				Yes	Yes			Classification	Balance	Not		No
Amasaki et al. [5]	2015	Data simplification	14(44)	14(44)	Java			Yes		Yes	Yes		Classification	F_1, AUC	All		Yes
Singh et al. [99]	2015	Prediction at design phase	7	7	C/C++/Java			Yes					Classification	AUC	All		Yes
Zhang et al. [126]	2015	Classifier ensemble	10	10	Java						Yes		Both	NofB20, F_1	All		Yes

(Continued)

Table 2. Continued

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Satin et al. [96]	2015	Impact of classifier	1270	1270	C/Pascal/C++/ Java/C#			Yes			Yes		Classification	AUC	Not		No
Cao et al. [13]	2015	Transfer defect learning	8	8	C/Java			Yes	Yes	Yes			Classifcation	F_1	All		Yes
Altinger et al. [3]	2015	Adoption in automotive industry	2	2	C			Yes	Yes	Yes			Classifcation	F_1	Not		No
He et al. [30]	2015	Transfer defect learning	11	11	Java			Yes		Yes	Yes	Yes	Classifcation	F_1	Not		No
Panichella et al. [82]	2014	Classifier ensemble	10	10	Java					Yes			Both	AUC, AUCEC	All		Yes
Mizuno et al. [66]	2014	Text-based prediction	8(28)	8(28)	Java			Yes					Classification	F_1	All		Yes
He et al. [27]	2014	Imbalanced feature sets	11	11	Java		Yes	Yes		Yes			Classification	F_1	All		Yes
He et al. [28]	2014	Training data simplification	10(34)	10(34)	Java			Yes		Yes			Classification	F_1, G_1	All		Yes
Ma et al. [60]	2014	Associative classification	9	9	C/C++/Java			Yes		Yes	Yes		Classification	AUC	Not		No
Peters et al. [84]	2013	Balancing privacy and utility	10	10	Java	Yes		Yes		Yes			Classification	G_1	All		Yes
Peters et al. [86]	2013	Training data selection	12(35)	18(21)	Java			Yes					Classification	G_1	All		Yes
Turhan et al. [107]	2013	Mixed project data	41(73)	41(73)	C/C++/Java			Yes		Yes			Classification	Balance	All		Yes
Singh et al. [100]	2013	Utility of OO metrics	2	2	C++/Java								Classification	F_1	Not		No
Moshtari et al. [72]	2013	Vulnerability predictors	5	5	C/C++/Java								Classification	F2, AUC	Not		No

(Continued)

Table 2. Continued

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Nam et al. [80]	2013	Transfer defect learning	8	8	Java			Yes		Yes			Classification	F_1	All		Yes
He et al. [31]	2013	Cross proprietary projects	10(34)	10(34)	Java			Yes	Yes	Yes	Yes		Classification	G_1	All		No
Ma et al. [61]	2012	Transfer defect learning	7	3	C/C++/Java			Yes		Yes			Classification	F_1, AUC,	All		Yes
Rahman et al. [90]	2012	Effort-aware evaluation	9(38)	9(38)	Java			Yes					Both	F_1, AUC, AUCEC	Not		No
Uchigaki et al. [108]	2012	Classifier ensemble	12	12	C/C++/Java			Yes		Yes			Classification	AUC	Partial		Yes
He et al. [32]	2012	Feasibility of cross projects	10(34)	10(34)	Java			Yes					Classification	F_1	All		Yes
Babic [7]	2012	Adaptive defect prediction	8	8	Java			Yes	Yes	Yes	Yes		Classification	F_1	Not		No
Yu et al. [119]	2012	Utility of complexity metrics	5	4	C						Yes		Classification	Recall, Precision	Yes		Yes
Rremraj et al. [88]	2011	Utility of network metrics	3	3	Java			Yes		Yes	Yes		Classification	F_1	Not		No
Liu et al. [59]	2010	Classifier ensemble	7	7	C/C++/Java			Yes					Classification	NECM	Partial		Yes
Jureczko et al. [43]	2010	Training data selection	38(92)	38(92)	Java			Yes			Yes		Ranking	E1(80)	All		No
Turhan et al. [104]	2010	Regularities in defect prediction	13	13	C/C++/Java			Yes	Yes	Yes			Classification	Balance	Partial		No
Turhan et al. [106]	2009	Applicability of cross companies	10	10	C/C++/Java			Yes					Classification	Balance	Not		No
Zimmer. et al. [129]	2009	Applicability of cross projects	12(28)	12(28)	C/C++/C#/Java			Yes					Classification	Recall, Precision	Not		No

(Continued)

Table 2. Continued

Study	Year	Topic	Project characteristics			Key modeling components (challenges) covered						Performance evaluation context				Against SSM?	Select for comp.?
			#Source projects (releases)	#Target projects (releases)	Languages in source and target projects	Privatize data	Homogenize features	Filter instances	Balance classes	Transform distributions	Select features	Target training data	Application scenario	Main performance indicators	Test data avail?		
Turhan et al. [105]	2009	Fine-grained prediction	10	25	C/C++/Java			Yes					Ranking	E2(70)	Not		No
Cruz et al. [10]	2009	Distribution transforma-tion	1	2	Java					Yes	Yes		Classification	Z*	Not		No
Khoshgoftaar et al.[48]	2009	Multi-dataset classifier ensemble	7	7	C/C++/Java			Yes		Yes			Classification	NECM	Partial		Yes
Watanabe et al. [110]	2008	Applicability of cross languages	2	2	C++/Java				Yes				Classification	Recall, Precision	Not		No
Nagappan et al. [74]	2006	Utility of complexity metrics	5	5	C++/C#					Yes	Yes		Ranking	S	Not		No
Nagappan et al. [73]	2006	Utility of pro-cess/product metrics	1	1	C/C++					Yes	Yes		Ranking	S	Not		No
Thongmak et al. [103]	2003	Utility of design metrics	1	1	N/A						Yes		Classification	Accuracy	Not		No
Briand et al. [9]	2002	Applicability of cross projects	2	2	Java						Yes		Both	Corr., Comp., CECM	Not	Yes	No

TL: transfer Learning, CIL: class imbalance learning, SSM: simple size model.
RCEC: represents the raw cost-effectiveness curve in a module-based Alberg diagram, in which the x-axis is the cumulative modules inspected and the y-axis is the cumulative defects found.
CECM: Cost-effective curve in a module-based Alberg diagram (the x-axis is the cumulative number of the modules selected from the module ranking and the y-axis is the cumulative number of defects found in the selected modules).

CPDP models were more cost-effective than simple module size models. However, the statistical significance and effect sizes were not examined. Currently, for most of the existing CPDP models, it is unclear whether they have a prediction performance superior to simple module size models.

These observations reveal that much effort has been devoted to developing supervised CPDP models. However, little effort has been devoted to examining whether they are superior to simple module size models. It is important for researchers and practitioners to know the answer to this problem. If the magnitude of the difference were trivial, then simple module size models would be preferred in practice due to the low building and application cost. To answer this problem, we next compare the prediction performance of the existing supervised CPDP models with simple module size models.

3 EXPERIMENTAL DESIGN

In this section, we first introduce the simple module size models under study. Then, we present the research questions relating simple module size models to the supervised CPDP models. After that, we describe the data analysis method used to investigate the research questions. Finally, we report the datasets used.

3.1 Simple Module Size Models

In this study, we leverage simple module size metrics such as SLOC in the target release to build simple module size models. As stated by Monden et al. [67], to adopt defect prediction models in industry, one needs to consider not only their prediction performance but also the significant cost required for metrics collection and modeling themselves. A recent investigation from Google developers shows that a prerequisite for deploying a defect prediction model in a large company such as Google is that it must be able to scale to large source repositories [56]. Therefore, our study only considers those simple models that have a low building cost, a low application cost, and a good scalability. More specifically, we take into account the following two simple module size models: ManualDown and ManualUp. For the simplicity of presentation, let m be a module in the target release, SizeMetric be a module size metric, and $R(m)$ be the predicted risk value of the module m. Formally, the ManualDown model is $R(m) = \text{SizeMetric}(m)$, while the ManualUp model is $R(m) = 1/\text{SizeMetric}(m)$. For a given target release, ManualDown considers a larger module as more defect-prone, as many studies report that a larger module tends to have more defects [65]. However, ManualUp considers a smaller module as more defect-prone, as recent studies argue that a smaller module is proportionally more defect-prone and hence should be inspected/tested first [49–51, 65].

Under ManualDown or ManualUp, it is possible that two modules have the same predicted risk values, i.e., they have a tied rank. In our study, if there is a tied rank according to the predicted risk values, the module with a lower defect count will be ranked higher. In this way, we will obtain simple module size models that have the "worst" predictive performance (theoretically). If the experimental results show that those "worst" simple module size models are competitive with the existing CPDP models, then we can safely conclude that, for practitioners, it would be better to apply simple module size models to predict defects in a target release. Note that, Canfora et al. used Trivial_{Inc} and Trivial_{Dec} as the baseline models to investigate the performance of their proposed CPDP models under the ranking scenario [12]. Conceptually, Trivial_{Inc} is the same as ManualUp, while Trivial_{Dec} is the same as ManualDown. However, there are two important differences in the implementation. First, Trivial_{Inc} and Trivial_{Dec} are applied to the z-score normalized module size data, while ManualUp and ManualDown are applied to the raw/unhandled module size data. Second, in Canfora et al.'s study, they did not report how to process the tied rank in Trivial_{Inc} and Trivial_{Dec}.

As described in Section 2.3, there are two categories of performance indicators for model evaluation: non-effort-aware and effort-aware. A model would have a better non-effort-aware (effort-aware) performance if the top ranked modules contain more modules that are defective (have a high defect density). As mentioned above, many studies observe that larger modules tend to have more defects but have a lower defect density [73, 74, 78, 103]. Therefore, ManualDown tends to outperform ManualUp under non-effort-aware indicators, while ManualUp tends to outperform ManualDown under effort-aware indicators. This means that, when investigating the predictive power of a CPDP model under a particular indicator, there is no need to use ManualDown and ManualUp as the baseline models at the same time. In our study, for a CPDP model, if the involved performance indicator is non-effort-aware, we use ManualDown as the baseline model for comparison; if the involved performance indicator is effort-aware, we use ManualUp as the baseline model. As shown in Table 1, all of the performance indicators in the classification scenario are non-effort-aware. In the ranking scenario, AUCEC, NofB20, PofB20, and E2(R) are effort-aware, while FPA, E1(R), and Prec@k are non-effort-aware. Therefore, in our study, for the classification scenario, only ManualDown is used as the baseline model. However, for the ranking scenario, ManualDown is used as the baseline model if the involved indicators are FPA, E1(R), and Prec@k, while ManualUp is used as the baseline model if the involved indicators are AUCEC, NofB20, PofB20, and E2(R).

Note that, in the classification scenario, all of the performance indicators except AUC are threshold-dependent. For ManualDown, there is a need to determine a classification threshold to compute these threshold-dependent indicators. In our study, we first rank the modules in a target release from the highest to lowest predicted risk value by ManualDown. Then, we classify the top $x\%$ modules in the ranking list as defective and the remaining modules as not defective. In the following, we use ManualDown($x\%$) to denote the ManualDown model with the classification threshold of $x\%$. For the simplicity of presentation, x is set to 50 in our study. We will discuss the influence of different classification thresholds on our conclusions in Section 4.

3.2 Research Questions

Our study aims to investigate the benefits of the existing supervised CPDP models w.r.t. simple module size models for defect prediction. To this end, our study will investigate the following two research questions.

RQ1: *(Classification performance comparison) How well do the existing supervised CPDP models perform compared against simple module size models when classifying defect-prone modules?*

RQ2: *(Ranking performance comparison) How well do the existing supervised CPDP models perform compared against simple module size models when ranking defect-prone modules?*

The purposes of RQ1 and RQ2 are to compare the existing supervised CPDP models against simple module size models under two prediction scenarios (i.e., the classification and ranking scenarios) to determine how well they predict defective modules. Since the existing supervised CPDP models leverage the defective or not defective label information to build the prediction models, we expect that they perform better than simple module size models. However, if the existing supervised CPDP models are only marginally better than or similar to simple module size models, it would be a good option for practitioners to apply simple module size model to predict defects in a target project. To the best of our knowledge, little is currently known on this topic. Our study attempts to fill this gap by conducting an in-depth comparison on the difference in the

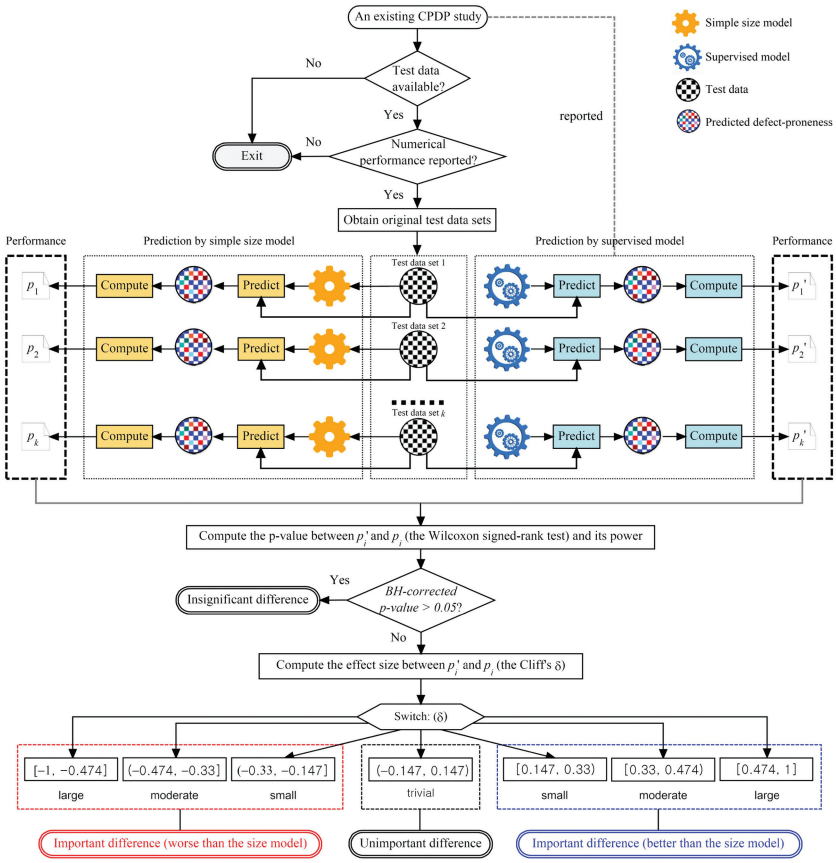

Fig. 3. The data analysis method for comparing an existing supervised CPDP model against the simple module size.

prediction performance between the existing supervised CPDP models and simple module size models.

3.3 Data Analysis Method

Figure 3 describes the data analysis method for RQ1 and RQ2 at a high level. For a given CPDP study, we first examine whether it uses publicly available test data and reports the prediction performance in numerical form. If both of the answers are yes, then we next investigate the prediction performance of the proposed supervised CPDP model compared with simple module size models. As described in Section 3.2, RQ1 focuses on classification performance, while RQ2 focuses on ranking performance. Without loss of generality, we assume that (1) p is the performance indicator under investigation and (2) the CPDP study uses k target releases to test the proposed supervised CPDP model.

To answer RQ1 and RQ2, we will employ the following analysis procedure:

Step 1: *obtain the test dataset t_i $(i = 1 \ldots k)$ used in the original CPDP study.* For the ith target release, if the original CPDP study uses a small number of modules as the target training modules, t_i consists of the corresponding complete labeled target release data excluding

the target training data.[3] Otherwise, t_i consists of the corresponding complete labeled target release data.

Step 2: *collect the reported prediction performance on t_i ($i = 1 \ldots k$) by the supervised CPDP model.* Let p'_i be the prediction performance of the supervised CPDP model on the test dataset t_i in the CPDP study. Consequently, we have k performance values for the supervised CPDP model.

Step 3: *collect the prediction performance p_i on t_i ($i = 1 \ldots k$) by the simple module size model.* For the ith test dataset t_i, we first generate the corresponding simple size model and then compute the prediction performance p_i on t_i. As a result, we have k performance values for the simple module size model.

Step 4: *determine the statistical significance of the difference between p'_i and p_i (using the Wilcoxon signed-rank test [113]) and the corresponding power.* The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to determine whether two matched samples are selected from populations having the same distribution. If the Wilcoxon's signed-rank test reports a p-value lower than 0.05, then the difference between p'_i and p_i is considered to be significantly different and otherwise not. Since multiple tests between the supervised CPDP and simple module size models may result in spurious statistically significant results, we use Benjamini-Hochberg correction of p-values (i.e., BH-corrected p-values) to control for false discovery [8]. In particular, we compute the power of the Wilcoxon signed-rank test. Informally, the power of a statistical test is the probability to detect a difference, if the difference actually exists [18]. By convention, when the power achieves 80% or more, we are confident about the insignificant difference reported by the Wilcoxon's signed-rank test. According to Reference [98], the power of the Wilcoxon signed-rank test is influenced by the statistical significance level α, the effect size measured by Cohen's d [22], the distribution of the difference between p'_i and p_i, and the sample size. The prominent distributions considered in the literature include normal, logistic, and Laplace [21, 98], which respectively correspond to standard-, lightly heavy-, and heavy-tailed distributions. In our study, we observe that the performance differences between most supervised CPDP models and simple module size models exhibit a heavy tail distribution. Therefore, for a given CPDP model, we first use the Anderson-Darling test [22] provided in the tool Easyfit 5.6[4] to examine whether the performance difference follows a Laplace or logistic distribution at the significance level of 0.05. After that, we use the tool G*Power 3.1.9.2 [21] to compute the power of the Wilcoxon signed-rank test. By this step, we can determine whether the performance difference is statistically significant with a high power.

Step 5: *determine the practical importance of the difference between p'_i and p_i ($i = 1 \ldots k$).* If step 4 reports that the difference between p'_i and p_i ($i = 1 \ldots k$) is statistically significant, then we next use the Cliff's δ [91], a non-parametric effect size measure, to quantify the magnitude of the difference. By convention [91], the magnitude of the difference is considered trivial for $|\delta| < 0.147$, small for $0.147 \leq |\delta| < 0.33$, moderate for $0.33 \leq |\delta| < 0.474$, and large for $|\delta| \geq 0.474$. From the viewpoint of practical use, if $|\delta| < 0.147$, then there is an ignorable difference of the prediction performance between the supervised CPDP model and the simple module size model. In the other cases, the difference is important ($\delta \leq -0.147$

[3] In this case, we use the algorithm shared by the original CPDP study to generate the target training data. This ensures that we can obtain the same test data as used in the original CPDP study.

[4] http://www.mathwave.com/easyfit-distribution-fitting.html.

indicates that the supervised CPDP model is worse than the simple module size model, while $\delta \geq 0.147$ indicates the opposite).

By the above analysis procedure, we can know whether the existing supervised CPDP models have a superior classification (RQ1) or ranking performance (RQ2) than the simple module size models.

As shown in the last column in Table 2, for 42 CPDP studies, the test datasets are publicly available. Of the 42 CPDP studies, four studies [5, 26, 28, 47] report only the median or mean prediction performance on the test datasets (rather than the prediction performance on individual test datasets). In addition, one study [119] only has four target releases. Consequently, we cannot apply the Anderson-Darling test to examine the distribution of the performance difference (as it is required that the data sample must contain at least five values). Therefore, for each of the 5 studies [5, 26, 28, 47, 119], we are unable to apply the procedure described in Figure 3 to analyze whether there is a statistically significant or important difference between the proposed supervised CPDP model and the simple module size model. Section A in the supplementary materials provides a comparison of their median or mean prediction performance. For each of the remaining 37 CPDP studies, we will apply the analysis procedure described in Figure 3 to investigate the benefits of the proposed CPDP models w.r.t. simple module size models and report the experimental results in Section 4.

3.4 Datasets

Table 3 summarizes the test datasets used in the 42 CPDP studies under investigation. The first column reports the name of the group. The second to seventh columns, respectively, report the target project (release), the project type, the programming language used, the project description, the number of instances contained (one instance corresponding to one module), and the percentage of defective instances. Note that, for the target project data in the NASA group, the CPDP studies use either the original version or the preprocessed version [56]. Therefore, for most NASA projects, there are multiple values under the sixth and seventh columns. The last column reports the size metric used to build the simple module size models in our study.

From Table 3, we have the following observations. First, the target projects cover both open-source and closed-source projects. Of the 58 target projects,[5] 35 projects are closed-source projects. Furthermore, of the 35 closed-source projects, 17 projects are academic projects, while the other 18 projects are industrial projects. Second, Java is the dominant development language in the target projects. Of the 58 target projects, 42 projects were developed using Java. Of the remaining 16 projects, 4 projects were developed using C++, while 12 projects were developed using C. Third, the target projects cover wide-ranging domains and functions. On the one hand, the target projects are from different domains, including real-time software, embedded software, and management software. On the other hand, the functions implemented in the target projects are wide-ranging, including integration development environment, text editor, and text/image-processing library. Fourth, the number of instances varies greatly across the target projects. The maximum number of instances is 25,157, while the minimum number of instances is only 10. In particular, it appears that C projects tend to have more instances than Java projects, as they have a finer module granularity (function vs. class). Fifth, the defect data show a highly imbalanced distribution in the target projects. For most projects, the percentage of defective instances is well below 20%. This is especially true for C projects, two of which even have less than 1% defective instances. Finally, the size metric is simple and has a low data collection cost. The

[5]The Lucene project appears in both the AEEEM group and the Metrics Repo group. Therefore, it is counted only once here.

Table 3. Details of the Test Datas Involved in the Existing Supervised Cross-Project Defect Prediction Study

Group	Target project (release)	Type	Lang.	Description	#Instances	%Defective	Module size metric
NASA	CM1	CSS	C	A spacecraft instrument	327-998	9.50-12.84%	LOC_EXECUTABLE (The number of lines of executable code for a module)
	JM1			A real time C project	10878-25157	19.32-20.28%	
	KC1		C++	A storage management system for ground data	2107-2109	15.42-15.46%	
	KC2			A storage management system for ground data	520-522	20.38-20.50%	
	KC3		Java	A storage management system for ground data	194-588	9.39-45.24%	
	MC1		C++	A combustion experiment	9466-13630	0.72-2.14%	
	MC2			A video guidance system	125-161	32.30-35.20%	
	MW1		C	A zero gravity experiment related to combustion	253-770	7.69-10.67%	
	PC1			A flight software from an earth orbiting satellite	705-1109	6.87-8.65%	
	PC2			A dynamic simulator for attitude control systems	5589	0.41%	
	PC3			A flight software from an earth orbiting satellite	1077-1563	10.24-12.44%	
	PC4			A flight software from an earth orbiting satellite	1458	12.21%	
Softlab	AR1			Embedded software	121	7.44%	executable_loc(The number of lines of executable code for a module)
	AR3			Dishwasher	63	12.70%	
	AR4			Refrigerator	107	18.69%	
	AR5			Washing machine	36	22.22%	
	AR6			Embedded software	101	14.85%	
AEEEM	Equinox 3.4	OSS	Java	An OSGi R4 core framework implementation	324	39.81%	ck_oo_numberOfLinesOfCode (Number of lines of code)
	JDT Core 3.4			The Java infrastructure of the Java IDE	997	20.66%	
	Lucene 2.4.0			A text search engine library	691	9.26%	
	Mylyn 3.1			A task management plugin	1862	13.16%	
	PDE UI 3.4.1			A plug-in development environment	1497	13.96%	
Relink	Apache httpd 2.0			An open-source HTTP server	194	46.91-50.52%	CountLineCode (Number of lines of code)
	OpenIntents Safe			An open intents library	56	28.57-39.29%	
	ZXing 1.6			A barcode image-processing library	399	20.80-29.57%	
Metrics Repo	Ant 1.3, 1.4, 1.5, 1.6, 1.7			A build management system	125-745	10.92-26.21%	
	Camel 1.0, 1.2, 1.4, 1.6			A versatile integration framework	339-965	3.83-35.53%	
	ckjm			A tool for collecting Chidamber and Kemerer metrics	10	50%	
	Ivy 1.1, 1.4, 2.0			A dependency manager	111-352	6.64-56.76%	
	JEdit 3.2, 4.0, 4.1, 4.2, 4.3			A text editor	272-492	2.24-33.09%	
	Log4j 1.0, 1.1, 1.2			A logging utility	109-205	25.19-92.20%	
	Lucene 20, 2.2, 2.4			A text search engine library	195-340	46.67-59.71%	
	PBeans 1.0, 2.0			An object/relational database mapping framework	26-51	19.61-76.92%	
	Poi 1.5, 2.0, 2.5, 3.0			API for Office Open XML standards	237-442	11.78-64.42%	
	Redaktor			A decentralized content management system	176	15.34%	
	Synapse 1.0, 1.1, 1.2			A high-performance Enterprise Service Bus	157-256	10.19-33.59%	

(Continued)

Table 3. Continued

Group	Target project (release)	Type	Lang.	Description	#Instances	%Defective	Module size metric
	Tomcat 6.0			A Web server	858	8.97%	
	Velocity 1.4, 1.5, 1.6			A template language engine	196-229	34.06-75.00%	
	Xalan 2.4, 2.5, 2.6, 2.7			An XSLT processor	723-909	15.21-98.79%	
	Xerces 1.0, 1.2, 1.3, 1.4			An XML processor	162-588	15.23-74.32%	
	prop6			An industrial projects in the insurance domain	660	10.00%	
	arc				234	11.54%	loc(Number of lines of code)
	berek				43	37.21%	
	e-learning				64	7.81%	
	forrest 0.6, 0.7, 0.8				6-32	6.25-17.24%	
	intercafe				27	14.81%	
	kalkulator				27	22.22%	
	nieruchomosci				27	37.04%	
	pdftranslator	CSS		Academic software projects developed by 8th	33	45.45%	
	serapion			or 9th semester computer science students	45	20.00%	
	skarbonka				45	20.00%	
	sklebagd				20	60.00%	
	systemdata				65	13.85%	
	szybkafucha				25	56.00%	
	termoproject				42	30.95%	
	workflow				39	51.28%	
	wspomaganiepi				18	66.67%	
	zuzel				29	44.83%	
Eclipse	Eclipse 2.0, 2.1, 3.0	OSS		An integrated development environment (IDE)	6729-10593	10.83-14.80%	TLOC (Total lines of code)

CSS: Closed-source project; OSS: Open-source project.

size metric is either the number of lines of executable code or the number of lines of source code, excluding blank lines and comment lines. In practice, it is easy to collect such a simple size metric.

The above observations reveal that cross-project defect prediction has been investigated in diverse contexts, including different development paradigms, different programming languages, and different project characteristics. In all of the target project (release) datasets, the simple size metric is available. This enables us to use the same test datasets to make a fair comparison in defect prediction performance between the existing supervised CPDP studies and simple module size models.

4 EXPERIMENTAL RESULTS

In this section, we report in detail the experimental results from the comparison of the existing supervised CPDP models with the size models.[6] In Section 4.1, we report the results on classification performance. In Section 4.2, we report the results on ranking performance.

4.1 RQ1: Classification Performance Comparison

Table 4 reports the results on the improvement in classification performance of the supervised CPDP models over the simple module size model. Here, the simple module size model denotes ManualDown. When computing the threshold-dependent performance indicators such as F_1 and G_1, we use 50% as the classification threshold (i.e., the model is ManualDown(50%)). The first column lists the CPDP studies under investigation. The 2nd column lists for each study the name of the proposed supervised CPDP model used for comparison (Section D in the supplementary materials provides the short descriptions for these models). The 3rd column lists the number of source-target project combinations from which the prediction performances of the CPDP and ManualDown models are collected. The 4th column lists the performance indicators for comparison. The 5th and 6th columns, respectively, report for each model the mean performance and the corresponding standard deviation (in parentheses). The 7th, 8th, and 9th columns report the descriptive statistics for the performance difference between the CPDP model and ManualDown, including the mean, median, and standard deviation. The 10th and 11th report the results from the statistical test, including the BH-corrected p-value and the power. The last two columns report the effect size. In particular, there are three types of backgrounds: *cyan*, *red*, and *white*. A cyan background indicates that the corresponding supervised CPDP model performs significantly better than ManualDown, a red background indicates that the corresponding supervised CPDP model performs significantly worse than ManualDown, and a white background indicates that there is no significant difference.

Table 5 summarizes the win/tie/loss results when comparing the supervised CPDP models against ManualDown in the classification scenario. The row "Number of comparisons" reports the number of times that the supervised CPDP models obtain a better, similar, and worse prediction according to the statistical significance (the second column) or according to the practical importance of the performance difference (the third column). As shown in Table 4, there are 42 performance comparisons involved. Therefore, the total number of win, tie, and loss times is 42. The row "Number of studies" reports the number of studies that the supervised CPDP models obtain a better, equal, and worse prediction according to the statistical significance (the second column) or the importance of the performance difference (the third column). In our context, a CPDP model is said to "win" in a study if it performs better than ManualDown according to at least one indicator and performs similarly according to the remaining indicators. In contrast, a CPDP model is said to "loss" in a study if it performs worse than ManualDown according to at least one indicator and performs similarly according to the remaining indicators. Furthermore, a CPDP model is said to

[6] A replication kit can be downloaded via https://zenodo.org/record/1209483.

Table 4. The Improvement in Classification Effectiveness of Supervised CPDP Models over ManualDown

Study	The CPDP model	N	Indicator	Model performance		Difference			Statistical test		Effect size	
				CPDP	ManualDown	mean	median	sd	p-value*	power	cliff's δ	Size
Zhang et al. [122]	MT+	18	F_1	0.448(0.170)	0.484(0.178)	-0.036	-0.021	0.146	0.728	0.237	-0.136	Trivial
			AUC	0.722(0.092)	0.749(0.065)	-0.027	-0.024	0.080	0.525	0.395	-0.148	Small
Stuckman et al. [101]	Metric+CFA	6	F_1	0.294(0.245)	0.269(0.258)	0.025	-0.001	0.052	1.000	0.249	<-0.001	Trivial
Aarti et al. [1]	ANN	19	F_1	0.266(0.130)	0.382(0.134)	-0.116	-0.086	0.177	0.045	0.920	-0.457	Moderate
Herbold et al. [37]	EM	79	F_1	0.419(0.176)	0.464(0.201)	-0.045	-0.022	0.136	0.025	0.947	-0.141	Trivial
			AUC	0.643(0.093)	0.734(0.111)	-0.090	-0.093	0.099	<0.001	1.000	-0.507	Large
Jing et al. [42]	SSTCA+ISDA	8	F2	0.661(0.034)	0.582(0.093)	0.080	0.068	0.068	0.030	0.959	0.563	Large
			AUC	0.796(0.022)	0.760(0.062)	0.036	0.049	0.065	0.525	0.417	0.438	Moderate
Krishna et al. [52]	Bellwether	19	ED	0.386(0.073)	0.343(0.073)	0.043	0.024	0.093	0.228	0.660	0.277	Small
Xia et al. [116]	HYDRA	29	F_1	0.544(0.223)	0.515(0.164)	0.029	0.008	0.114	0.728	0.382	0.062	Trivial
Zhang et al. [123]	Universal	5	AUC	0.741(0.060)	0.725(0.056)	0.016	-0.013	0.051	1.000	0.112	0.120	Trivial
Wang et al. [109]	DBN	22	F_1	0.568(0.151)	0.534(0.131)	0.033	-0.032	0.168	1.000	0.199	0.103	Trivial
Ryu et al. [93]	VCB-SVM	9	AUC	0.768(0.128)	0.788(0.074)	-0.020	0.005	0.066	1.000	0.179	0	Trivial
Ryu et al. [92]	MONBNN	13	Balance	0.621(0.073)	0.629(0.088)	-0.008	-0.014	0.039	0.620	0.133	-0.189	Small
			AUC	0.686(0.105)	0.701(0.133)	-0.015	-0.016	0.062	0.526	0.179	-0.172	Small
Hosseini et al. [39]	GIS	13	F_1	0.484(0.284)	0.486(0.216)	-0.002	-0.024	0.099	1.000	0.051	0.018	Trivial
			G_2	0.543(0.241)	0.526(0.184)	0.017	-0.014	0.085	0.938	0.137	0.065	Trivial
Cheng et al. [17]	CCT-SVM	32	F_1	0.556(0.156)	0.400(0.135)	0.156	0.113	0.197	<0.001	1.000	0.541	Large
			AUC	0.741(0.062)	0.763(0.068)	-0.022	-0.019	0.076	0.369	0.506	-0.179	Small
Kaur et al. [45]	POP	16	AUC	0.762(0.087)	0.747(0.086)	0.015	0.022	0.034	0.341	0.549	0.102	Trivial
Catal et al. [14]	Threshold	6	AUC	0.685(0.055)	0.755(0.064)	-0.070	-0.068	0.037	0.094	0.998	-0.611	Large
He et al. [29]	TDSelector	15	AUC	0.720(0.080)	0.721(0.110)	-0.002	-0.003	0.059	0.938	0.053	-0.102	Trivial
Ryu et al. [94]	HISNN	6	Balance	0.669(0.041)	0.646(0.039)	0.023	0.036	0.035	0.369	0.427	0.389	Moderate
Peters et al. [85]	LACE2	10	G_1	0.583(0.051)	0.662(0.048)	-0.078	-0.066	0.039	0.010	1.000	-0.760	Large
Jing et al. [41]	CCA+	74	F_1	0.592(0.170)	0.418(0.156)	0.174	0.141	0.188	<0.001	1.000	0.553	Large
Nam et al. [79]	HDP	28	AUC	0.711(0.108)	0.713(0.114)	-0.003	-0.001	0.026	0.648	0.102	-0.036	Trivial
Singh et al. [99]	NB	42	AUC	0.701(0.099)	0.758(0.083)	-0.052	-0.030	0.095	0.002	0.954	-0.286	Small
Zhang et al. [126]	Max	10	F_1	0.413(0.115)	0.399(0.136)	0.014	0.038	0.120	0.588	0.069	0.100	Trivial
Cao et al. [13]	TCANN	26	F_1	0.451(0.186)	0.450(0.178)	0.001	<0.001	0.035	1.000	0.054	-0.009	Trivial
Panichella et al. [82]	CODEP(BN)	10	AUC	0.861(0.072)	0.730(0.111)	0.131	0.088	0.105	0.010	0.994	0.820	Large
Mizuno et al. [66]	Text	6	F_1	0.463(0.223)	0.503(0.255)	-0.040	0.002	0.120	1.000	0.145	-0.167	Small
He et al. [27]	IFS(tca)	19	F_1	0.475(0.104)	0.469(0.172)	0.007	0.004	0.164	1.000	0.055	0.058	Trivial
Peters et al. [84]	LACE1(m40)	10	G_1	0.598(0.101)	0.604(0.104)	-0.006	-0.010	0.064	1.000	0.062	-0.060	Trivial
Peters et al. [86]	Peter-filter	21	G_1	0.694(0.265)	0.667(0.172)	0.028	0.061	0.342	0.622	0.072	0.263	Small
Turhan et al. [107]	Mixed	32	Balance	0.623(0.105)	0.638(0.063)	-0.015	0.010	0.110	1.000	0.157	-0.021	Trivial
Nam et al. [80]	TCA+	26	F_1	0.454(0.151)	0.450(0.178)	0.004	-0.013	0.061	1.000	0.066	0.030	Trivial
Ma et al. [61]	TNB	9	F_1	0.376(0.172)	0.370(0.152)	0.006	0.003	0.031	0.938	0.102	0.012	Trivial
			AUC	0.699(0.076)	0.788(0.074)	-0.089	-0.076	0.031	0.018	1.000	-0.630	Large
Uchigaki et al. [108]	Ensemble	20	AUC	0.701(0.058)	0.724(0.064)	-0.024	-0.033	0.019	0.004	1.000	-0.430	Moderate
He et al. [32]	DT	34	F_1	0.627(0.156)	0.490(0.172)	0.137	0.121	0.067	<0.001	1.000	0.420	Moderate
Liu et al. [59]	GP(V-V)	18	NECM	0.741(0.153)	0.730(0.102)	0.011	-0.005	0.069	1.000	0.093	-0.022	Trivial
Khoshgoftaar et al.[48]	MLMD	18	NECM	0.862(0.214)	0.730(0.102)	0.133	0.139	0.167	0.031	0.978	0.380	Moderate

*: Benjamini-Hochberg-corrected p-value.
BN: Bayes Network.

Table 5. Win/Tie/Loss in the Classification Prediction Scenario

	Statistical significance	Practical importance
Number of comparisons	5/29/8	5/30/7
Number of studies	5/23/7	5/23/7

"tie" in a study if it performs similarly according to all of the indicators or if it performs better according to one indicator but performs worse according to another indicator. As shown in Table 4, there are 35 studies involved. Therefore, the total number of win, tie, and loss studies is 35.

From Table 5, we make the following important observations. First, the supervised CPDP models do not show a classification performance superior to ManualDown(50%) in more than 85% (37/42 = 88.10%) of the performance comparisons, regardless of whether the statistical significance or the practical importance of the prediction performance difference is taken into account. Second, the supervised CPDP models do not show a classification performance superior to ManualDown(50%) in more than 85% (30/35 = 85.71%) of the investigated CPDP studies, regardless of whether the statistical significance or the practical importance of the prediction performance difference is taken

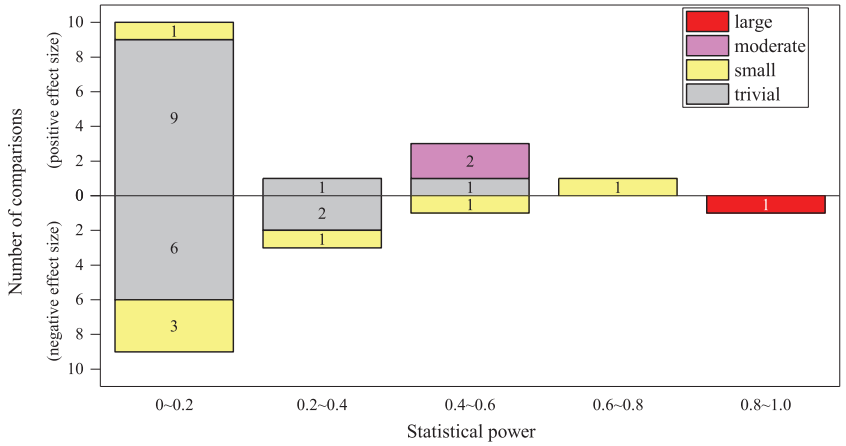

Fig. 4. The distribution of the effect sizes in 29 "tie" comparisons over the statistical power.

into account. Overall, the results indicate that, when classifying defect-prone modules, most of the supervised CPDP models proposed in the literature to date are indeed not superior to the ManualDown model built on the target projects.

Note that, the above analysis does not take into account the statistical power of the Wilcoxon signed-rank test. According to the value "5/29/8" in Table 5, 29 comparisons do not report a significant difference in classification performance between the supervised CPDP models and ManualDown. However, as shown in Table 4, almost all of the 29 "tie" comparisons have a low statistical power. This is a possible reason why the Wilcoxon signed-rank test is unable to detect a significant difference in these comparisons. To examine the influence of the statistical power on our results, we use Figure 4 to describe the distribution of the effect sizes in these 29 comparisons over the statistical power. The upper y-axis is the number of positive effect sizes, while the lower y-axis is the number of negative effect sizes. For example, of the 29 comparison, 19 comparisons have a power within the interval [0, 0.2), 10 producing positive effect sizes (nine trivial and one small) and 9 producing negative effect sizes (six trivial and three small). As can be seen, of the 29 effect sizes, only four are non-trivial and positive effect sizes. This means that, in most cases (i.e., 25/29 = 86.21%), ManualDown is still a good alternative to the supervised CPDP models even if the Wilcoxon signed-rank test has a high power enough to detect a significant difference in their classification performance.

4.1.1 A Close Examination of Winning Supervised CPDP Models. From Table 4, we can see that the following five supervised CPDP models are significantly better than ManualDown(50%): He et al.'s DT [32], Panichella et al.'s CODEP(Bayes) [82], Jing et al.'s CCA+ [41], Cheng et al.'s CCT-SVM [17], and Jing et al.'s SSTCA+ISDA [42]. Table 6 provides a summary for these five CPDP models, including the original publication (i.e., the publication that a model is originally proposed), the topic, and the key modeling components covered. In particular, for each supervised CPDP model, the last column lists the external validation studies found in the literature (i.e., the publications that externally validate the model). For example, for CODEP originally proposed in Panichella et al.'s study, there are four external validation studies (i.e., [35, 42, 116, 126]). Overall, it seems that instance filtering and/or distribution transformation play an important role in these five CPDP models.

For each of the five supervised CPDP model, given the promising results reported in the original study, a natural problem raised is to what extent the findings can be generalized to the population

Table 6. A Comparison of Five Supervised CPDP Models That Perform Better Than ManualDown in Classification

Original study	Model	Topic	Key modeling components (challenges) covered				External validation study
			Homogenize features	Filter instances	Balance classes	Transform distributions	
Jing et al. [42]	SSTCA+ISDA	Class imbalance learning		Yes	Yes	Yes	[57]
Cheng et al. [17]	CCT-SVM	Heterogeneous prediction	Yes	Yes		Yes	[57]
Jing et al. [41]	CCA+	Heterogeneous prediction	Yes	Yes		Yes	[17, 57, 58]
Panichella et al. [82]	CODEP(BN)	Classifier ensemble				Yes	[35, 42, 116, 126]
He et al. [32]	DT	Feasibility of cross projects		Yes			

beyond the sample projects under study (i.e., external validity). The external validation studies usually use different source and target projects compared with those used in the original study, thus providing us a good opportunity to evaluate the external validity of a CPDP model. If the CPDP model has a high external validity, then it is important to understand why they perform well. This would be helpful for developing better CPDP models in the future. To this end, we next take a close examination of the above five supervised CPDP models. More specifically, for each supervised CPDP model having external validations, we use the performance reported in these external validation studies to investigate whether it still outperforms ManualDown. To facilitate the discussion, we will analyze them in order from the earliest to the latest publication time.

He et al.'s DT [32]. For a given target project, He et al. used the combination of source projects providing the best prediction result as the training dataset. In their experiment, decision table (DT) was the best performing modeling technique. Their purpose was to determine whether cross-project data could provide acceptable prediction results. The premise of obtaining such a best prediction, however, is to know the defect information in the target project during model building. In practice, such information is not available at that time. In other words, practitioners are indeed unable to build such a CPDP model.

Panichella et al.'s CODEP [82]. Panichella et al. proposed CODEP (COmbined DEfect Predictor) to combine different base classifiers to improve the detection of defect-prone modules. More specifically, CODEP is a classifier that is trained on the outcomes of different base classifiers, which are trained on the training data. In their study, Panichella et al. used six base classifiers, including Logistic Regression (LR), Bayes Network (BN), Radial Basis Function Network (RBF), Multi-layer Perceptron (MLP), Alternating Decision Trees (ADTree), and Decision Table (DT). In particular, Panichella et al. proposed using LR and BN to conduct the combination and the results show that CODEP(BN) has a higher AUC than CODEP(LR). According to Table 4, CODEP(BN) has a significantly higher AUC than ManualDown and the effect size is large. By Google scholar, we find that CODEP has been externally validated in four recent studies. Jing et al. [42], Xia et al. [116], and Zhang et al. [126] used CODEP as a baseline model to evaluate their newly proposed CPDP models, while Herbold et al. [35] evaluated the performance of CODEP in a benchmark study for CPDP. Since each external validation study used publically available datasets, we were able to compute the classification performance of ManualDown on the same target projects used to evaluate CODEP. Therefore, we can make a comparison on the prediction performance between CODEP and ManualDown. Table 7 summarizes the results on the comparison obtained

Table 7. Classification Performance Comparison: CODEP vs. ManualDown

Study	The CPDP model	N	Indicator	Model performance		Difference			Statistical test		Effect size	
				CPDP	ManualDown	mean	median	sd	p-value*	power	cliff's δ	Size
Panichella et al. [82]	CODEP(BN)	10	AUC	0.861(0.072)	0.730(0.111)	0.131	0.088	0.105	0.004	0.994	0.820	Large
Jing et al. [42]	CODEP	8	F2	0.415(0.071)	0.582(0.093)	-0.167	-0.197	0.068	0.018	1.000	-0.875	Large
			AUC	0.635(0.035)	0.760(0.062)	-0.125	-0.136	0.067	0.012	1.000	-1.000	Large
Xia et al. [116]	CODEP(LR)	29	F_1	0.417(0.116)	0.515(0.164)	-0.098	-0.055	0.129	<0.001	0.998	-0.400	Moderate
Zhang et al. [126]	CODEP(LR)	10	F_1	0.301(0.131)	0.399(0.136)	-0.098	-0.032	0.212	0.261	0.385	-0.340	Moderate
Herbold et al. [35]	CODEP(BN)	86	F_1	0.360(0.169)	0.475(0.204)	-0.115	-0.089	0.174	<0.001	1.000	-0.339	Moderate
			G_1	0.467(0.197)	0.646(0.085)	-0.178	-0.152	0.192	<0.001	1.000	-0.573	Large
			MCC	0.238(0.151)	0.255(0.169)	-0.017	-0.013	0.130	0.268	0.324	-0.082	Trivial
			AUC	0.617(0.080)	0.732(0.113)	-0.115	-0.111	0.093	<0.001	1.000	-0.626	Large

*: Benjamini-Hochberg-corrected p-value.

Table 8. Classification Performance Comparison: CCA+ vs. ManualDown

Study	N	Indicator	Model performance		Difference			Statistical test		Effect size	
			CPDP	ManualDown	mean	median	sd	p-value*	power	cliff's δ	Size
Jing et al. [41]	74	F_1	0.592(0.170)	0.418(0.156)	0.174	0.141	0.188	<0.001	1.000	0.553	Large
Cheng et al. [17]	32	F_1	0.535(0.161)	0.400(0.135)	0.135	0.102	0.201	<0.001	0.996	0.480	Large
		AUC	0.719(0.062)	0.763(0.068)	-0.044	-0.037	0.078	0.009	0.969	-0.346	Moderate
Li et al. [57]	30	AUC	0.641(0.062)	0.749(0.072)	-0.108	-0.109	0.075	<0.001	1.000	-0.739	Large
Li et al. [58]	28	F2	0.482(0.112)	0.517(0.144)	-0.035	-0.058	0.097	0.074	0.622	-0.154	Small
		G_3	0.635(0.051)	0.642(0.079)	-0.007	-0.021	0.073	0.399	0.099	-0.230	Small
		AUC	0.669(0.058)	0.713(0.114)	-0.043	-0.059	0.094	0.009	0.830	-0.421	Moderate

*: Benjamini-Hochberg-corrected p-value.

from Panichella et al.'s original study and from the four external validation studies. Note that Jing et al. [42] did not report whether BN or LR was used to combine six based classifiers in CODEP. From Table 7, we can see that, in contrast to Panichella et al.'s results, the overall results from the four external validation studies show that CODEP is indeed inferior to ManualDown.

Jing et al.'s CCA+ [41]. Jing et al. proposed CCA+ to predict defect-prone modules in a target project using source projects with heterogeneous metric sets. They first defined a unified metric representation (UMR) for source and target projects by combining the common metrics, source-project-specific metrics, target-project-specific metrics, and an appropriate number of zeros. Then, they applied Canonical Correlation Analysis (CCA) to determine a transformation of the UMR that maximizes the correlation between source and target project data. Recently, Cheng et al. [17] and Li et al. [57, 58] reported the classification performance of CCA+ under different experimental settings. Table 8 summarizes the results on the comparison between CCA+ and ManualDown obtained from Jing et al.'s original study and from the three validation studies. As can be seen, the result obtained from Jing et al.'s study [41] shows that CCA+ is superior to ManualDown according to F_1, which is confirmed by the result obtained from Cheng et al.'s study [17]. However, according to the other performance indicators, the results obtained from all the three external validation studies show that CCA+ is similar to or even inferior to ManualDown. More importantly, although Jing et al. [41] and Cheng et al. [17] showed that CCA+ had a good classification performance in terms of F_1, the premise was to use appropriate source projects as the training data. However, they do not provide an automatic method to determine such source projects. Therefore, for a given target project, in practice, it is unknown how to select appropriate source projects to ensure that the resulting CCA+ model has a high F_1.

Cheng et al.'s CCT-SVM [17]. Based on CCA+, Cheng et al. proposed CCT-SVM (Cost-sensitive Correlation Transfer Support Vector Machine) to deal with heterogeneous cross-project defect prediction. According to Table 4, compared with ManualDown, CCT-SVM has a higher F_1 (significant) but a lower AUC (not significant). In Reference [57], Li et al. reported that CCT-SVM had a

Table 9. Classification Performance Comparison: SSTCA+ISDA vs. ManualDown

Study	N	Indicator	Model performance		Difference			Statistical test		Effect size	
			CPDP	ManualDown	mean	median	sd	p-value*	power	cliff's δ	Size
Jing et al. [42]	74	F2	0.661(0.034)	0.582(0.093)	0.080	0.068	0.068	0.008	0.959	0.563	Large
		AUC	0.796(0.022)	0.760(0.062)	0.036	0.049	0.065	0.250	0.417	0.438	Moderate
Li et al. [57]	30	AUC	0.659(0.064)	0.749(0.072)	-0.089	-0.086	0.032	<0.001	1.000	-0.633	Large

*: Benjamini-Hochberg-corrected p-value.

mean AUC of 0.591 on 30 target projects. However, on the same 30 target projects, ManualDown has a mean AUC of 0.749, thus exhibiting a 26.73% improvement over CCT-SVM. Similar to CCA+, in practice, it is unknown how to select appropriate source projects to ensure that the resulting CCT-SVM model has a high F_1.

Jing et al.'s SSTCA+ISDA [42]. Jing et al. proposed SSTCA+ISDA to deal with the class-imbalance problem in CPDP They first used Semi-Supervised Transfer Component Analysis (SSTCA) to make the distributions of source and target data consistent. Then, they applied ISDA (Improved Subclass Discriminant Analysis) to the transformed data to learn features with favorable prediction ability for CPDP. Table 9 summarizes the results on the comparison between SSTCA+ISDA and ManualDown obtained from Jing et al.'s study [42] and from Li et al.'s study [57]. As can be seen, compared with ManualDown, SSTCA+ISDA has a lower AUC in Li et al.'s study but has a higher F2 in Jing et al.'s study. In their study, Jing et al. used 16 projects from four groups to conduct the CPDP experiment: five from the NASA dataset, five from the AEEEM dataset, three from the Relink dataset, and three from the SOFTLAB dataset. Within each group of projects, they conducted all one-to-one and others-to-one CPDP. The former predicts defects in one target project using the data from only one source project, while the latter predicts defects in one target project using the combined data of all the other projects. In this sense, in practice, for a given target project, applying SSTCA+ISDA in the way of one-to-one or others-to-one combination could achieve a higher F2 than ManualDown.

Overall, the above examination reveals that the five winning supervised CPDP models in Table 4 can be classified into three categories. The first category consists of He et al.'s DT [32], Jing et al.'s CCA+ [41], and Cheng et al.'s CCT-SVM [17]. Compared with ManualDown, practitioners may benefit from them if appropriate source projects are used to build the CPDP models. Before applying them in practice, there is a need to develop an automatic method to determine the appropriate source projects for a given target project. However, such an automatic method is currently unavailable in the literature. The second category consists of only Panichella et al.'s CODEP [82]. Although the result obtained from Panichella et al.'s original study shows that CODEP is superior to ManualDown, the results obtained from four external validation studies are negative. Therefore, the benefit of applying CODEP in practice is questionable. The third category consists of only Jing et al.'s SSTCA+ISDA [42]. The current results show that practitioners may benefit from SSTCA+ISDA in terms of F2 compared with ManualDown. However, the evidence is only from one individual study and hence a further investigation on its prediction performance is necessary in the future.

4.1.2 Influence of Classification Threshold. In the above analysis, the classification threshold for the ManualDown model is set to 50%. However, many performance indicators (such as F_1 and G_1) depend on the threshold, which may have an unknown influence on our conclusion. To examine this unknown influence, we next vary the threshold from 10% to 90% to conduct the comparison between the supervised CPDP model and the ManualDown model.

Figure 5 reports how the number (left y-axis) and percentage (right y-axis) of "tie"+"loss" comparisons vary with the classification threshold. In each group, the left column corresponds to the

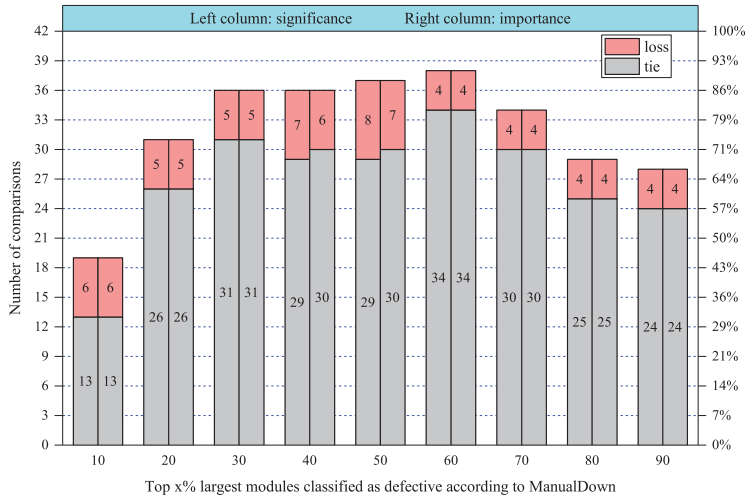

Fig. 5. The influence of classification threshold on the distribution of "tie"+"loss" comparisons.

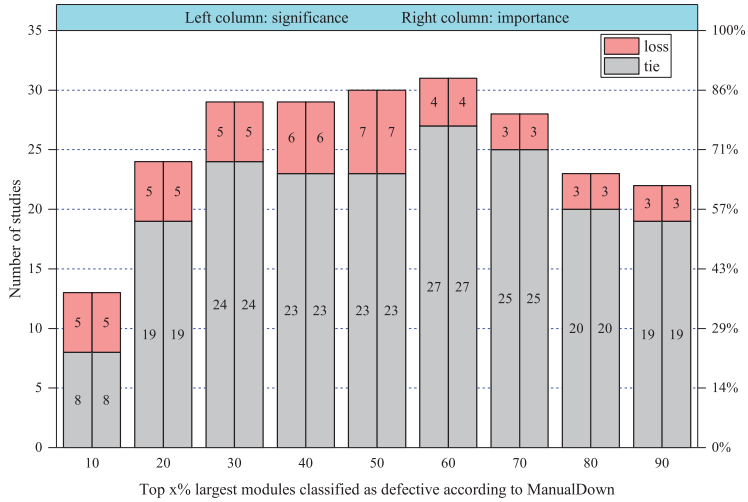

Fig. 6. The influence of classification threshold on the distribution of "tie"+"loss" studies.

"tie"+"loss" results based on the statistical significance of the prediction performance difference, while the right column corresponds to the "tie"+"loss" results based on the practical importance of the prediction performance difference. For example, under the threshold 40%, the left column indicates that there are 7 "loss" comparisons and 29 "tie" comparisons according to the statistical significance, while the right column indicates that there are 6 "loss" comparisons and 30 "tie" comparisons according to the practical importance. In both cases, the supervised CPDP models do not outperform than ManualDown(40%) in 86% ((7+29) / 42 = 85.71%, see the right y-axis) of the performance comparisons. Figure 6 reports how the number (left y-axis) and percentage (right y-axis) of "tie"+"loss" studies vary with the classification threshold. The core observation from Figure 5 and Figure 6 is that the results are similar when the classification threshold varies between 30% and 70%.

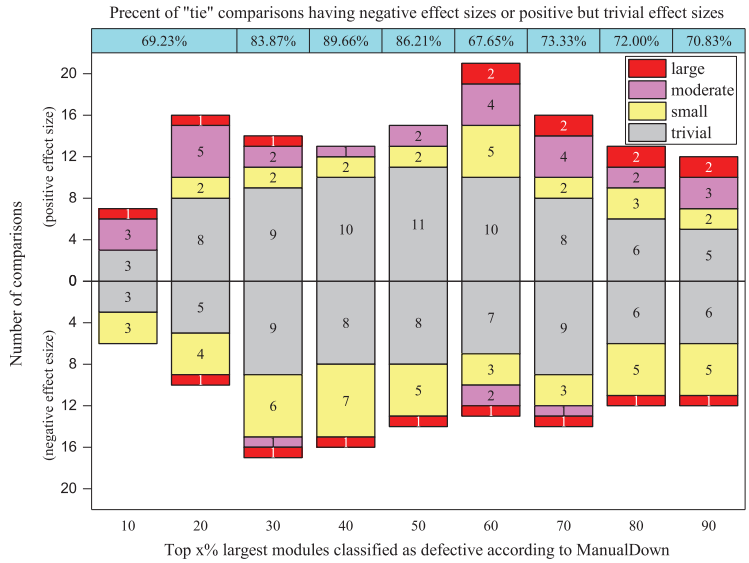

Fig. 7. The distribution of the effect sizes in "tie" comparisons over classification threshold.

From Figure 5, we can see that, under each classification threshold, many comparisons report an insignificant difference in classification performance between the supervised CPDP models and ManualDown. For example, 29 comparisons report an insignificant difference under the threshold of 40%, while 34 comparisons report an insignificant difference under the threshold of 60%. As shown in the third column in Table 4, the sample size is small in these comparisons (the largest sample size is 79). If the effect size is not large enough, then the Wilcoxon signed-rank test will have a low power and hence may fail to detect a significant difference. In this context, it is necessary to examine the direction and magnitude of effect sizes for insignificant comparisons. This will help to understand to what extent ManualDown is competitive to the supervised CPDP models in practice. Figure 7 reports the distribution of the effect sizes in "tie" comparisons when the classification threshold varies from 10% to 90%. The first row reports the percentage of comparisons having Negative effect sizes or Positive but Trivial effect sizes (NPT) under each classification threshold. For example, under the threshold 40%, 16 comparisons have negative effect sizes, while 10 comparisons have positive but trivial effect sizes. Therefore, the percentage of NPT under the threshold 40% is 89.66% (i.e., 26/29). We can observe that, for most thresholds, the percentage of NPT is above 70%. In particular, for the threshold 50%, the percentage of NPT is above 85%. This means that ManualDown would still be competitive to the supervised CPDP models even if the Wilcoxon signed-rank test were able to detect a significant difference in their classification performance.

When combining the results from Figure 5, Figure 6, and Figure 7, we find that the classification threshold may have an influence on the results of "tie"+"loss" comparisons. However, it appears that the influence is not large when the threshold varies between 30% and 70%. In particular, we observe that ManualDown(50%) performs well compared with the other ManualDown (x%) models where $x \neq 50$.

4.2 RQ2: Ranking Performance Comparison

Table 10 reports the results on the improvement in ranking performance of the supervised CPDP models over the simple module size model. Note that, the simple module size model denotes

Table 10. Improvement in Ranking Effectiveness of Supervised CPDP Models over Simple Size Models

Study	the CPDP model	N	Indicator	Model performance		Difference			Statistical test		Effect size	
				CPDP	ManualUp	mean	median	sd	p-value*	power	cliff's δ	Size
Xia et al. [116]	HYDRA	29	PofB20	0.330(0.146)	0.481(0.192)	-0.150	-0.109	0.157	<0.001	1.000	-0.467	Moderate
You et al. [118]	ROCPDP	39	Prec@10	0.382(0.179)	0.621(0.256)	-0.238	-0.200	0.263	<0.001	1.000	-0.563	Large
Canfora et al. [12]	MODEP(LR)	10	AUCEC	0.836(0.129)	0.759(0.158)	0.077	0.085	0.044	0.023	1.000	0.280	Small
Zhang et al. [126]	Bagging(J48)	10	NofB20	40.6(33.964)	37.7(31.457)	2.900	-2.000	15.481	1.000	0.104	-0.030	Trivial
Panichella et al. [82]	CODEP(LR)	10	AUCEC	0.541(0.079)	0.718(0.158)	-0.177	-0.111	0.224	0.046	0.814	-0.740	Large

*: Benjamini-Hochberg-corrected p-value.

Table 11. Win/tie/loss in the Ranking Prediction Scenario

	Statistical significance	Practical importance
Number of comparisons	1/1/3	1/1/3
Number of studies	1/1/3	1/1/3

Table 12. Ranking Performance Comparison: MODEP vs. ManualUp

Study	the CPDP model	N	Indicator	Model performance		Difference			Statistical test		Effect size	
				CPDP	ManualUp	mean	median	sd	p-value*	power	cliff's δ	Size
Canfora et al. [12]	MODEP(LR)	10	AUCEC	0.836(0.129)	0.759(0.158)	0.077	0.085	0.044	0.028	1.000	0.280	Small
Xia et al. [116]	MODEP(LR)	29	PofB20	0.191(0.109)	0.481(0.192)	-0.290	-0.295	0.243	<0.001	1.000	-0.772	Large
Chen et al. [16]	MULTI-M	30	ACC	0.730(0.078)	0.747(0.073)	-0.017	-0.007	0.119	0.487	0.148	-0.211	Small
			Popt	0.889(0.045)	0.899(0.043)	-0.009	-0.007	0.070	0.487	0.143	-0.289	Small

*: Benjamini-Hochberg-corrected p-value.

ManualDown for You et al. [118], as Prec@10 is a non-effort performance indicator. In the other cases, the simple module size model denotes ManualUp. As can be seen, in four of the five investigated studies, the supervised CPDP models perform similarly to or even significantly worse than the simple module size model in the ranking scenario.

Table 11 summarizes the win/tie/loss results when comparing the supervised CPDP models against the simple module size model in the ranking scenario. As can be seen, both the total number of comparison times and the total number of studies are 5. From Table 11, we can observe that the supervised CPDP models do not show a ranking performance superior to the simple module size model in 80% (4/5 = 80%) of the performance comparisons, regardless of whether the statistical significance or the practical importance of the prediction performance difference is taken into account. This is also true for the investigated CPDP studies. Overall, the results indicate that, when ranking defect-prone modules, most of the existing supervised CPDP models are indeed not superior to the simple module size model built on the target projects.

4.2.1 A Close Examination of Winning Supervised CPDP Models. From Table 10, we can see that Canfora et al.'s MODEP(LR) is significantly better than the simple module size model ManualUp. Canfora et al. used NSGA-II, a multi-objective genetic algorithm, to estimate the coefficients of a logistic regression model that minimize cost and maximize performance. The cost was measured by Lines Of Code (LOC) that must be reviewed, while the performance was measured in recall.

By Google scholar, we find that MODEP(LR) has been externally validated in two recent studies under the ranking scenario. Xia et al. [116] used MODEP(LR) as a baseline model to evaluate their newly proposed CPDP model, while Chen et al. [16] applied MODEP(LR) to change-level defect prediction. For each of the two studies, we computed the ranking performance of ManualUp on the same target projects. Consequently, we can make a comparison on the ranking performance between MODEP(LR) and ManualUp. Table 12 summarizes the results on the comparison obtained from Canfora et al.'s original study and from the two recent studies. Note that Chen et al. [16] named MODEP(LR) as MULTI and reported the median performance of ACC and Popt on multiple

independent runs. ACC denotes the recall of buggy changes when using 20% of the entire efforts, which is indeed equivalent to PofB20. Popt is the AUCEC normalized to the optimal and worst models. As can be seen, in contrast to Canfora et al.'s results, the overall results from the two external validation studies show that MODEP is indeed inferior to or similar to ManualUp.

5 COMPARISON WITH RELATED WORK

In this section, we compare simple module size models with related work, including the supervised CPDP models investigated in a recent benchmark study and two state-of-the-art unsupervised models.

5.1 Results from Cross-Project Defect Prediction Benchmark

Recently, Herbold et al. [35, 36] pointed out that, although many CPDP models had been proposed, it was unclear which one performed best. The main reason for this is that the evaluations are based on diverse experimental setups, including using different performance metrics and using different datasets. To attack this problem, Herbold et al. conducted a CPDP benchmark study [35, 36].

In the benchmark study, Herbold et al. replicated 24 CPDP approaches proposed between 2008 and 2015 [5, 10, 11, 26, 31, 33, 47, 48, 59, 61, 63, 78, 80, 82–85, 93, 94, 106, 108, 110, 126, 129]. Of the 24 approaches, 17 approaches [5, 10, 26, 31, 33, 47, 48, 61, 78, 80, 83–85, 94, 106, 110, 129] were indeed a treatment for the data, i.e., they did not propose a classifier. As shown in Table 13, Herbold et al. applied six classifiers (DT, LR, NB, RF, NET, and SVM) to each of these 17 approaches to build the CPDP models. For the approach Menzies11 [63], in addition to the above six classifiers, Herbold et al. also applied the WHICH algorithm, which was used in the original publication, to build the CPDP models. For the remaining six approaches [11, 59, 82, 93, 108, 126], Herbold et al. used the classifiers proposed in the original publications to build the CPDP models. In addition, Herbold et al. included the following four baseline approaches in the benchmark study: ALL (use all data for training without any data treatment), CV (10×10 cross-validations, a WPDP baseline), RANDOM (randomly classifies modules as defective), and FIX (classifies all modules as defective). For ALL and CV, Herbold et al. applied six classifier (DT, LR, NB, RF, NET, and SVM) to build the CPDP models.

Herbold et al. evaluated the investigated CPDP models on 86 datasets (i.e., 86 target releases, see Table 15 in Section B in the supplementary materials for details) from five groups: 62 from JURECZKO [43], 12 from NASA MDP [97], 5 from AEEEM [19], 4 from NETGENE [38], and 3 from RELINK [115]. They used AUC, F_1, G_1, and MCC to evaluate the performance of a model to predict defects in a target release. In particular, they published a replication kit [34], including the used datasets, all the scripts, and the raw results. This enables us to compare the prediction performance of simple module size models with their benchmarked CPDP models.

Classification performance comparison. In our experiment, we used the module size metric shown in Table 15 in Section B in the supplementary materials to build the simple module size model ManualDown. Figures 8–11, respectively, report the distributions of the improvement in F_1, G_1, MCC, and AUC of Herbold et al.'s benchmarked CPDP models over ManualDown. Due to the large number of results, we only list the best classifier for each CPDP approach. For example, we only list NB for Turhan09, because it outperformed DT, LR, RF, NET, and SVM. In each figure, a blue box-plot indicates "significantly better," a red box-plot indicates "significantly worse," and a black box-plot indicates "no significant difference" (according to the Benjamini-Hochberg-corrected p-value returned by the Wilcoxon signed-rank test). In particular, a box-plot filled with the gray color indicates there is a large effect size according to Cliff's δ. Note that, for Nam13, Herbold et al. only reported the results for 3 datasets, as they were unable to apply it to the other

Table 13. The CPDP Models Investigated in Herbold et al.'s Benchmark Study

Approach	Classifier	CPDP model
Liu10 [59]	Genetic programming (GP)	Liu10-GP
Uchigaki12 [108]	Logistic ensemble (LE)	Uchigaki12-LE
Canfora13 [11]	Multi-objective deffect predictor (MODEP)	Canfora13-MODEP
Panichella14 [82]	Combined defect predictor (CODEP) with logistic regression (CODEP-LR), CODEP with a Bayesian network (CODEP-BN)	Panichella14-CODEP-LR, Panichella14-CODEP-BN
Ryu16 [93]	Value-cognitive boosting with support vector machine (VCBSVM)	Ryu16-VCBSVM
YZhang15 [126]	Average voting (AVGVOTE), maximum voting (MAXVOTE), bagging with a C4.5 decision tree (BAG-DT), bagging with Naïve Bayes (BAG-NB), boosting with a C4.5 decision tree (BOOST-DT), boosting with Naïve Bayes (BOOST-NB)	YZhang15-AVGVOTE, YZhang15-MAXVOTE, YZhang15-BAG-DT, YZhang15-BAG-NB, YZhang15-BOOST-DT, Zhang15-BOOST-NB
Koshgoftaar09 [48]	C4.5 Decision Tree (DT), Logistic Regression (LR), Naïve Bayes (NB), Random Forest (RF), RBF Network (NET), Support Vector Machine (SVM)	Koshgoftaar09-DT, ..., Koshgoftaar09-SVM
Watanabe08 [110]		Watanabe08-DT, ..., Watanabe08-SVM
Turhan09 [106]		Turhan09-DT, ..., Turhan09-SVM
Zimmermann09 [129]		Zimmermann09-DT, ..., Zimmermann09-SVM
CamargoCruz09 [10]		CamargoCruz09-DT, ..., CamargoCruz09-SVM
Ma12 [61]		Ma12-DT, ..., Ma12-SVM
Peters12 [83]		Peters12-DT, ..., Peters12-SVM
Peters13 [84]		Peters13-DT, ..., Peters13-SVM
Herbold13 [33]		Herbold13-DT, ..., Herbold13-SVM
ZHe13 [31]		ZHe13-DT, ..., ZHe13-SVM
Nam13 [80]		Nam13-DT, ..., Nam13-SVM
PHe15 [26]		PHe15-DT, ..., PHe15-SVM
Peters15 [85]		Peters15-DT, ..., Peters15-SVM
Kawata15 [47]		Kawata15-DT, ..., Kawata15-SVM
Amasaki15 [5]		Amasaki15-DT, ..., Amasaki15-SVM
Ryu15 [94]		Ryu15-DT, ..., Ryu15-SVM
Nam15 [78]		Nam15-DT, ..., Nam15-SVM
Menzies11 [63]	DT, LR, NB, RF, NET, SVM, WHICH	Menzies11-DT, ..., Menzies11-WHICH

Note that: In Herbold et al.'s study [35], "Ryu16" was called "Ryu14" and "Koshgoftaar09" was called "Koshgoftaar08".

83 datasets due to the ultra-high resource consumption or runtime. As can be seen, according to F_1 and G_1, none of Herbold et al.'s benchmarked CPDP models is significantly better than ManualDown. In contrast, ManualDown performs significantly better than most of Herbold et al.'s benchmarked CPDP models (the effects sizes are large for many CPDP models). According to MCC, ManualDown is significantly worse than CV-RF (a WPDP baseline), Peters15-NB, Turhan09-NB, and YZhang15-MAXVOTE. However, the effect sizes are only trivial for the latter three CPDP models. According to AUC, ManualDown is only significantly worse than CV-RF with a small effect size. When combining the results from F_1, G_1, MCC, and AUC, we can conclude that the simple module size model is very competitive with Herbold et al.'s benchmarked CPDP models.

Ranking performance comparison. Herbold et al. [35, 36] reported that, overall, according to F_1, G_1, MCC, and AUC, CamargoCruz09-NB performed the best among all the investigated models. Next, we examine how well CamargoCruz09-NB performs in the ranking scenario when compared with a simple module size model. To this end, we used the module size metric shown in

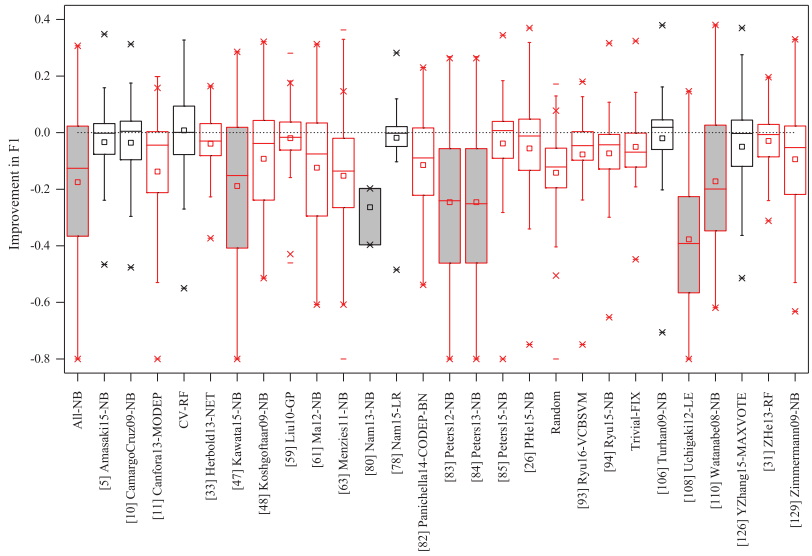

Fig. 8. The improvement in F_1 of Herbold et al.'s benchmarked CPDP models compared against the simple module size model ManualDown(50%).

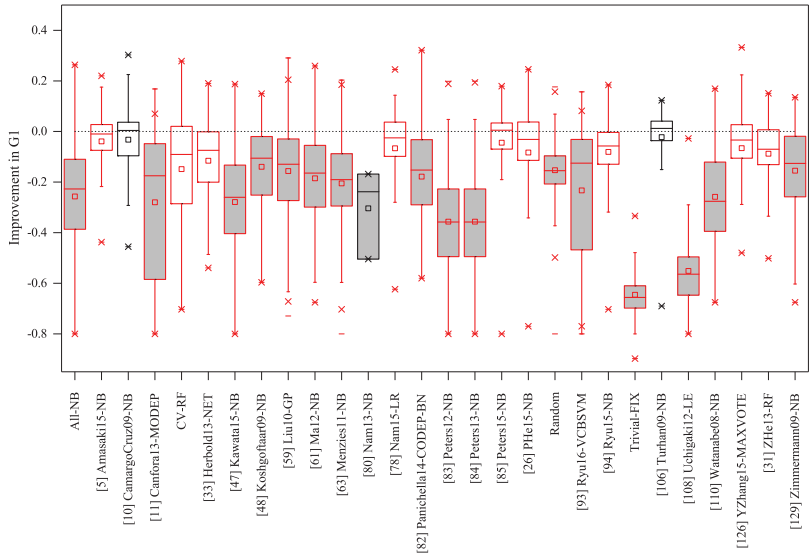

Fig. 9. The improvement in G_1 of Herbold et al.'s benchmarked CPDP models compared against the simple module size model ManualDown(50%).

Table 15 in Section B in the supplementary materials to build the simple module size model ManualUp. Figure 12 reports the distributions of the improvement in recall of CamargoCruz09-NB over ManualUp on the 86 target releases. The x-axis represents the code size in percentage of the top modules selected from the module ranking (relative to the total target project size), while the y-axis represents the percentage of defects contained in the selected modules. As can be seen, CamargoCruz09-NB is significantly worse than ManualUp, regardless of which code percentage

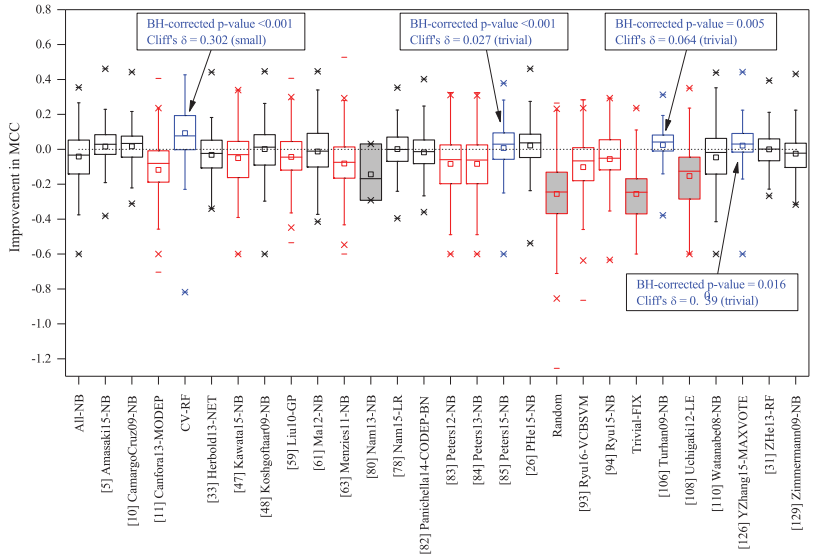

Fig. 10. The improvement in MCC of Herbold et al.'s benchmarked CPDP models compared against the simple module size model ManualDown(50%).

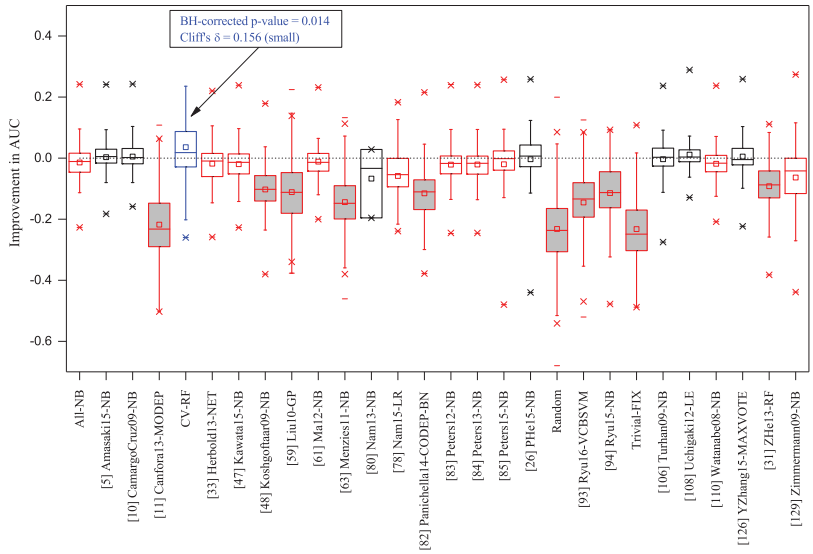

Fig. 11. The improvement in AUC of Herbold et al.'s benchmarked CPDP models compared against the simple module size model ManualDown.

is considered. In particular, for each code percentage, the magnitude of their difference is large in terms of Cliff's δ.

5.2 Results from the State-of-the-Art Unsupervised Models

In nature, the size model is a type of unsupervised model, as it does not employ any defect label information to build the model. Recently, researchers proposed a number of unsupervised models

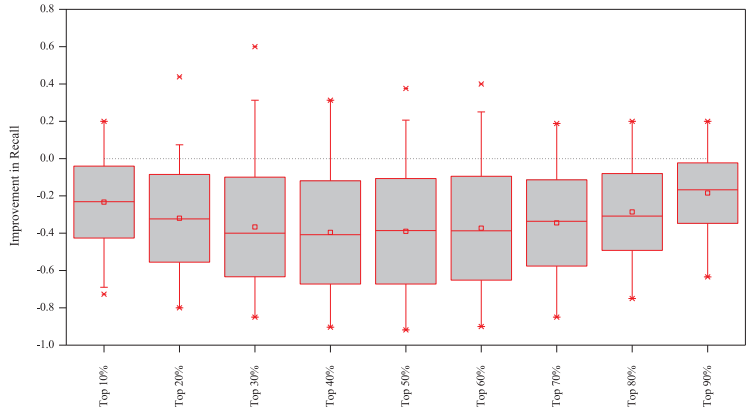

Fig. 12. The improvement in Recall of CamargoCruz09-NB over ManualUp when inspecting the top modules in the module ranking that constitute $x\%$ of the total code size.

and reported that they had a promising performance compared with the supervised CPDP models [78, 124]. Consequently, an interesting question raised is how well these newly proposed unsupervised models are when compared with the simple module size model. In Section 5.2.1, we compare the Spectral Clustering (SC) model [124] with the size model. In Section 5.2.2, we compare the Clustering-LAbeling (CLA) model [78] with the size model.

5.2.1 The SC Model vs. the Size Model. Zhang et al. proposed a connectivity-based unsupervised classification model called spectral clustering (SC) model [124]. For a given target project, the SC model predicts defective modules by the following steps:

(1) Construct a weighted graph for the target project. In this graph, nodes represent modules, edges represent the connection between modules, and weights represent module similarity. For two modules, their similarity is the dot product between the corresponding two z-score normalized metric vectors.
(2) Apply the spectral clustering algorithm to the graph to separate the modules into two non-overlapped clusters. According to Reference [129], "this algorithm partitions a graph into two subgraphs to gain high similarity within each subgraph while achieving low similarity across the two subgraphs."
(3) Label the modules in the cluster with larger average row sum of the normalized metrics as defective and the others as not defective. According to Reference [124], the reason is that "For most metrics, software entities containing defects generally have larger values than software entities without defects."

In particular, Zhang et al. [124] provided the R implementation of the SC model, consisting of only 17 lines of code. This enables other researchers to validate the SC model. Zhang et al. used the data of 26 projects from three groups (i.e., AEEEM [97], NASA [36], JURECZKO [43]) to compare the prediction performance of the SC model against five CPDP supervised models (i.e., random forest, naïve Bayes, logistic regression, decision tree, and logistic model tree). To evaluate the prediction performance of a supervised CPDP model on a given target project, Zhang et al. first trained the CPDP model using each of the remaining projects in the same group. Then, they reported the average AUC value of these models on the target project. In their experiment,

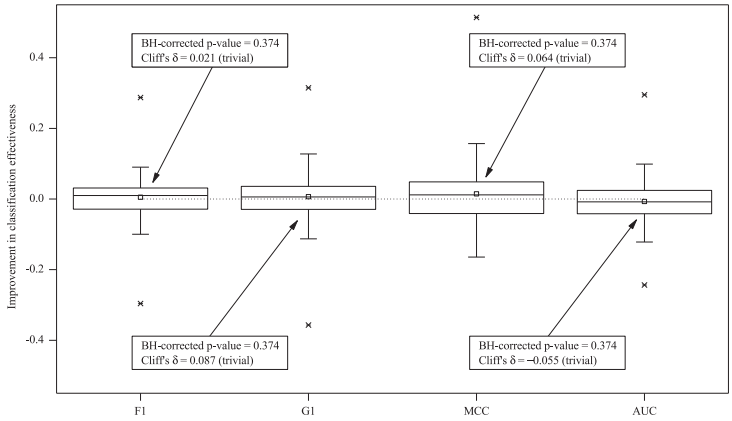

Fig. 13. The improvement in classification performance of Zhang et al.'s SC model compared against the simple module size model ManualDown.

Zhang et al. observed that the SC model performed better than all of the five supervised CPDP models.

To conduct a comprehensive comparison of the SC and simple module size models, we extended Zhang et al.'s experiment to include more datasets and more performance evaluation metrics. In our experiment, we used the 86 datasets shown in Table 15 in Section B in the supplementary materials to conduct the experiment. On the one hand, for each dataset, we used the corresponding size metric shown in Table 15 to build the ManualDown model. On the other hand, for each dataset, we first filtered out those metrics having a value of "NA" or having a constant value, as the SC algorithm was unable to process them. After that, we used the filtered data to build the SC model.[7] In addition to AUC, we also included F_1, G_1, and MCC for model evaluation. This allows us to compare the prediction performance of SC and ManualDown in a more comprehensive way.

Figure 13 describes the distributions of the improvement in F_1, G_1, MCC, and AUC of Zhang et al.'s SC model over ManualDown. For each performance metric, we also report the BH-corrected p-value and Cliff's δ. As can be seen, according to the BH-corrected p-value, the improvement of the SC model over ManualDown is not statistically significant, regardless of which performance metric is considered. Furthermore, according to Cliff's δ, the magnitude of their difference is trivial, regardless of which performance metric is considered. The above results indicate that the simple module size model is very competitive with Zhang et al.'s SC model. This is especially true for practitioners when the time complexity for model building is considered.

5.2.2 The CLA Model vs. the Size Model. Nam et al. proposed an unsupervised defect prediction model CLA (Clustering-LAbeling instances) to conduct defect prediction on the target project in an automated manner [78]. For a given target project, the CLA model predicts defective modules by the following steps:

[7]Since the SC algorithm is based on matrix operation, its computational load rapidly increases with the size of the input dataset. For example, we found that it took more than 12h to build the SC model for the MDP PC5 dataset (17,001 modules and 38 metrics). Our experiment was conducted on a computer with the following configuration: CPU: Intel Xeon(R) E7-4820 2.00GHz with 32-core; memory: 75GB; hard disk: 300GB; operation system: Windows Server 2012.

(1) Group instances (i.e., modules) to clusters based on the number of metrics having large values. First, use the median value for each metric as the threshold to determine large metric values. Second, count for each instance the number (K) of metrics whose values are greater than the threshold values. Third, group those instances that have the same K value as a cluster.
(2) Label the instances in the clusters whose K values are larger than the median K value as defective and the others as not defective. First, use K to divide the clusters into two groups: a top half and a bottom half. Second, label the instances in the top half of the clusters as defective and the others not defective. The reason behind this is that defective instances tend to have high metric values and the instances in the top half of the clusters have more high metric values.

In nature, for each module, the associated K is used to decide the defective or not-defective label, i.e., it indicates the defect-proneness of the module. To facilitate external validation, Nam et al. shared the source code of the CLA model online [77].

Nam et al. used the data of 7 datasets from two groups (i.e., NETGENE and RELINK) to compare the prediction performance of the CLA model against three baseline models: one supervised CPDP model built with simple logistic regression (SL), one threshold-based unsupervised model (THD), and one expert-based unsupervised model (EXP). The THD model predicts a module as defective when any metric value of the module is greater than the designated values of a threshold vector. In their study, Nam et al. generated the threshold vector by maximizing the number of correctly predicted instances in each target project. The EXP model first uses the K-means clustering algorithm to group instances into clusters and then asks human experts to label whether a cluster is defective or not. For each dataset, Nam et al. applied the EXP model to generate 20 clusters. In particular, they labeled a cluster as defective if the number of defective instances was greater than that of not defective instances and otherwise as not defective (as human experts knew the actual labels of instances). To evaluate the performance of defect prediction, Nam et al. employed precision, recall, and F_1. Overall, the experimental results showed that the CLA model was superior to the THD model and was comparable to the SL and EXP models.

To conduct a comparison of the CLA and simple module size models, we extended Nam et al.'s experiment to include more datasets and more performance evaluation metrics. More specifically, we used the 86 datasets shown in Table 15 in Section B in the supplementary materials to conduct the experiment. On the one hand, for each dataset, we used the size metric shown in Table 15 to build the ManualDown model. On the other hand, for each dataset, we first filtered out those metrics having a value of "NA" and then used the filtered data to build the CLA model. In addition to F_1, we also included G_1, MCC, and AUC for model evaluation. Note that, when computing AUC, we used the K associated with each module as the predicted defect-proneness.

Figure 14 describes the distributions of the improvement in F_1, G_1, MCC, and AUC of Nam et al.'s CLA model over ManualDown. As can be seen, according to the BH-corrected p-value, there is no statistically significant difference between CLA and ManualDown. Furthermore, according to Cliff's δ, the magnitude of the difference between CLA and ManualDown is trivial, regardless of which performance metric is considered. The above results indicate that the simple module size model is very competitive with Nam et al.'s CLA model.

6 DISCUSSION

In previous sections, we show that simple module size models have a prediction performance competitive to the supervised CPDP models. However, one may argue that the supervised CPDP models have the advantage that they can be applied to an incomplete target project or even to

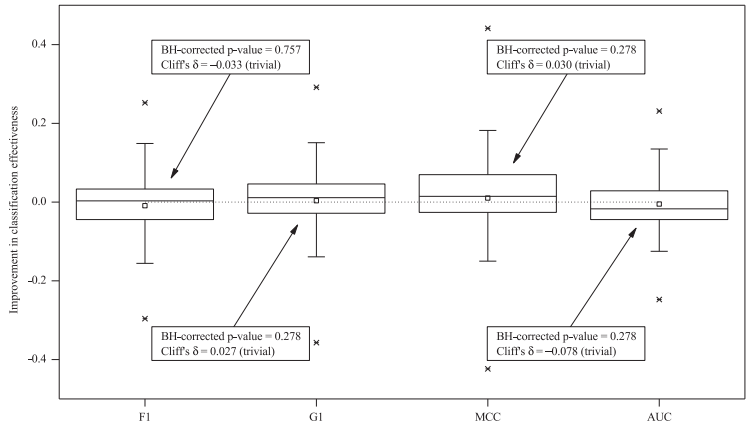

Fig. 14. The improvement in classification performance of Nam et al.'s CLA model compared against the simple module size model ManualDown.

a single module. How about simple module size models in these aspects? In this section, we not only discuss these problems (Section 6.1 and Section 6.2) but also summary the lessons learned for future CPDP research (Section 6.3).

6.1 Does the Application of a Size-Based Model Require the Availability of a Complete Target Project?

For a defect prediction model, an important problem that practitioners may concern is whether they can apply the model to an incomplete target project. As pointed out by an anonymous reviewer, if a developer has to wait for the completion of the target project to apply the model, it would be quite late in the development process and hence is not practical. As known, a (supervised) CPDP model can be applied to an incomplete target project. Is it true for a simple module size model? In our study, we used ManualDown and ManualUp, two simple module size models, to investigate the prediction performance of the existing supervised CPDP models. For a given target project, ManualDown was used to classify the modules (the top 50% defect-prone module and the bottom 50% not defect-prone), while ManualUp was used to rank the modules. In our experiment, we used complete target projects to evaluate ManualDown and ManualUp. In other words, ManualDown and ManualUp were applied to complete target projects in our study.

However, this does not necessarily mean that all of the simple module size models can only be applied in this way (i.e., requiring the availability of a complete target project). In the following, we show how to build a simple module size model that can be applied to an incomplete target project. In the literature, much effort has been devoted to determining the distribution of module size [4, 55, 69, 81, 117, 125]. In practice, we can use this distribution to build a simple module size model. For example, Zhang et al. [125] showed that the average module (i.e., Java class) size in terms of source lines of code is 114. Based on this information, we could build the following simple module size model (called SizeThreshold): if a module has a size larger than 114 SLOC, it is classified as defect-prone; otherwise, it is classified as not defect-prone. Given the size of a single module, SizeThreshold can predict whether it is defect-prone or not. Therefore, the application of SizeThreshold does not require a complete target project. We next use 74 Java datasets (62 from JURECZKO [43], 5 from AEEEM [19], 4 from NETGENE [38], and 3 from RELINK [115])

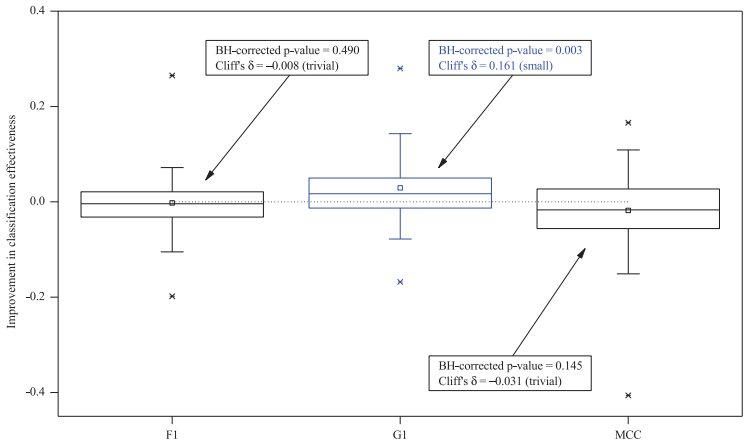

Fig. 15. The improvement in classification performance of ManualDown(50%) over SizeThreshold.

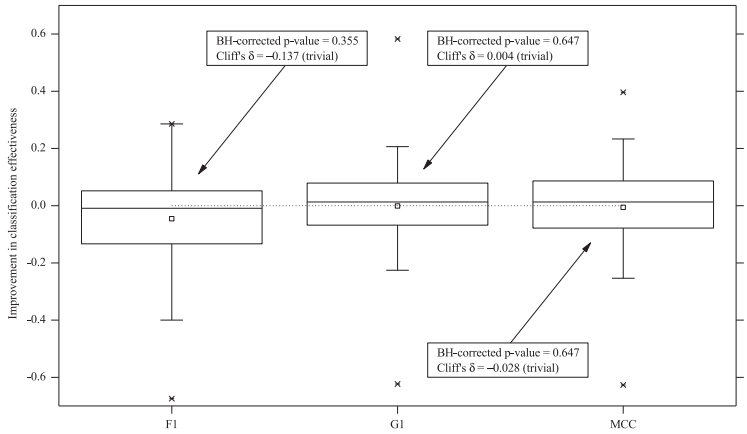

Fig. 16. The improvement in classification performance of CamargoCruz09-NB over SizeThreshold.

to investigate the performance of SizeThreshold when compared against ManualDown(50%) and CamargoCruz09-NB (the best performing supervised CPDP model reported in Reference [35]). Figure 15 describes the distributions of the improvement in F_1, G_1, and MCC of ManualDown(50%) over SizeThrehold. As can be seen, overall, SizeThreshold has a prediction performance similar to ManualDown(50%). Figure 16 describes the distributions of the improvement in F_1, G_1, and MCC of CamargoCruz09-NB over SizeThrehold. We can see that CamargoCruz09-NB does not outperform SizeThrehold. The key insight from the above example is that there exist simple (yet effective) size models whose application does not require the availability of a complete target project.[8]

[8]Note that there are two main reasons why we choose ManualDown and ManualUp, rather than the distribution-based size models such as SizeThrehold, as the baseline models in our article. First, they are easy-to-use models that have been used in many previous studies. Second, and more importantly, they have a good predictive ability, which is enough for our purpose to investigate the performance of the existing CPDP models in defect prediction.

6.2 Can a Size-Based Model Predict the Probability of a Single Module Being Defective?

For a defect prediction model, another important problem that practitioners may concern is whether they can apply the model to predict the probability of a single module being defective. If the answer is yes, then the predicted probability will be helpful for practitioners to allocate quality assurance resources effectively. At a first glance, it seems that, unlike a (supervised) CPDP model, a simple module size model is unable to make a prediction for a single module in a probabilistic way. This observation is true for ManualDown and ManualUp. However, this is not necessarily true for all of the simple module size models. In the following, we use SLOC as an example to show how to build a simple module size model that can predict the probability of a single module being defective. First, as described in Reference [54], we can use the empirical distribution of SLOC to determine two thresholds: Tmin and Tmax. A module is considered as low risk (i.e., a low probability of having defects) if its SLOC is less than Tmin and considered as high risk (i.e., a high probability of having defects) if its SLOC is larger than Tmax. Then, we can use binary logistic regression to build a size-based model:

$$\Pr(\text{SLOC}) = \frac{e^{a_0+a_1*\text{SLOC}}}{1+e^{a_0+a_1*\text{SLOC}}}.$$

Assume that

$$\Pr(\text{Tmin}) = \frac{e^{a_0+a_1*\text{Tmin}}}{1+e^{a_0+a_1*\text{Tmin}}} = p_0$$

$$\Pr(\text{Tmax}) = \frac{e^{a_0+a_1*\text{Tmax}}}{1+e^{a_0+a_1*\text{Tmax}}} = p_1.$$

In practice, p_0 (p_1) can be given by experience. Consequently, we can easily solve a_0 and a_1. For example, based on the PROMISE dataset, Lavazza et al. [54] reported that, for SLOC, according to "Adapted Distribution-Based" (ADB) method, Tmin = 62 and Tmax = 874, respectively, corresponding to the 16% and 84% quantiles of the SLOC distribution. Assume that $p_0 = 0.2$ and $p_1 = 0.8$. Then, we can easily conclude that $a_0 = -1.597994$ and $a_1 = 0.003414518$. Therefore, we have

$$\Pr(\text{SLOC}) = \frac{e^{-1.597994+0.003414518*\text{SLOC}}}{1+e^{-1.597994+0.003414518*\text{SLOC}}}.$$

We can see that this model can predict the probability of a single module having defects. The performance of this model in defect prediction depends on the availability of accurate size distribution information. Recently, many approaches have been proposed to determine the distribution of module size [4, 54, 55, 69, 81, 117, 125], which is out of the research scope of our article. The key insight from the above example is that it is feasible to build a simple module size model to predict the probability of a single module being defective.

6.3 What Lessons We Can Learn for Future Cross-Project Defect Prediction Research?

The results were very surprising for us, as we did not expect that simple module size models had a prediction performance comparable or even superior to most of the existing CPDP models. This informs that the real progress in CPDP is not being achieved as it might be envisaged in numerous CPDP publications. From our study, we learned three lessons for future cross-project defect prediction research.

Include simple module size models as the baseline models. Our community constantly updates the state-of-the-art by introducing "new" increasingly complex machine-learning techniques to build CPDP models. These "new" techniques are often compared against recently published techniques. The underlying assumption is that these "new" techniques are useful if they show

prediction performance superior to recently published techniques. Unfortunately, almost all of the existing CPDP models built with machine-learning techniques, including the "new" introduced and previously published, are not compared against simple module size models. In fact, simple module size models such as ManulDown and ManualUp perform well over a range of different project types [2, 49–51, 65, 102, 127, 128], which is further confirmed in our study. In particular, they have no parameters required tuning, are deterministic in their outcomes, and are easy to implement. In contrast, many CPDP models, especially those proposed recently, are based on complex modeling techniques that not only have a high computation cost but also involve a number of parameters needed to be carefully tuned. Furthermore, it is not a common practice for CPDP researchers to share their codes, where a tiny difference in the implementation may lead to a huge difference in the prediction performance of defect prediction. This imposes substantial barriers to apply them in practice, especially for large projects. In a future CPDP study, if prediction performance is the objective, one needs to show that the proposed CPDP model is superior to simple module size models. Otherwise, the motivation for introducing new CPDP models could not be well justified. Therefore, we suggest that ManulDown and ManualUp should be used as the baseline models in all future CPDP studies.

Deal with the CPDP challenges jointly rather than separately. As described in Figure 1, building a supervised CPDP model may involve many challenges, including privacy-preserving data sharing, metric set heterogeneity, irrelevant training data, class-imbalanced training data, metric data distribution heterogeneity, redundant features, and learning algorithm selection. In particular, there are complex interactions between these challenges. In other words, these challenges are not isolated but are closely related. For example, filtering out the ("irrelevant") training instances may have an influence on class imbalance, metric data distribution, and redundant features. However, most of the existing CPDP studies attack these challenges separately, as if they are completely independent. Furthermore, none of the existing CPDP studies attacks all the CPDP challenges jointly. This is a possible reason why many existing CPDP models do not perform well compared against simple module size models. Therefore, we recommend that future CPDP studies take into account these challenges simultaneously rather than separately when building CPDP models.

Use diverse datasets and performance indicators for holistic evaluation. Currently, more than 80 defect datasets are publicly available from the tera-PROMISE repository [64]. However, many CPDP studies only use a few datasets (rather than all the available datasets) for evaluating their proposed CPDP models. For example, of the 18 CPDP studies published in 2016 (see Table 2), more than 70% studies only used less than 20 datasets. Furthermore, a CPDP model might perform differently under different performance metrics. Although many performance metrics are available, most of the existing CPDP studies only employ one or two metrics for evaluation. Consequently, it is unknown how the proposed CPDP models perform on the unselected datasets and/or performance metrics. If the computational load is not a problem, then we recommend that future CPDP studies use all available datasets and diverse performance metrics for model evaluation. As highlighted in References [35], "such evaluations give a more holistic view on the capabilities of suggested approaches and techniques, from which we as a community will benefit in the long term."

Note that the above lessons are based on the premise that the goal of CPDP is to produce a high defect prediction performance. To the best of our knowledge, it appears that most (if not all) of the existing CPDP studies in the literature aim to develop a new CPDP model having a better prediction performance. Nonetheless, developers can use a CPDP model for other goals. For example, the goal is to examine whether the relationships between code (process) attributes and defects can transfer between projects or development contexts, i.e., the true value of a CPDP model is to understand whether the relationships in general exist. In this context, it is sufficient if a CPDP

model achieves reasonable performances in the target projects. In other words, there is no need to require that it outperforms the WPDP models such as simple module size models.

7 THREATS TO VALIDITY

We consider the most important threats to construct, internal, and external validity of our study. Construct validity is the degree to which the dependent and independent variables accurately measure the concept they purport to measure. Internal validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables. External validity is the degree to which the results of the research can be generalized to the population under study and other research settings.

7.1 Construct Validity

The dependent variable used in this study is a binary variable indicating whether a module is defective. Our study used the same datasets as used in the investigated cross-project defect prediction studies. These datasets were collected from either closed-source projects or open-source projects. For those datasets corresponding to open-source projects, the SZZ algorithm proposed by Sliwerski, Zimmermann, and Zeller was employed to discover the defective modules [131]. As a result, the discovered defective modules may be incomplete, which is a potential threat to the construct validity of the dependent variable. Indeed, this is an inherent problem to most, if not all, studies that discover defective modules by mining software repositories, not unique to us. Nonetheless, this threat needs to be eliminated by using complete defect data in the future work.

The independent variable used in this study is the module size metric. Our study used either lines of executable code or source lines of code, available in the datasets used by the investigated CPDP studies, as the module size metric to build the simple module size models. For their construct validity, previous research has investigated the degree to which they accurately measure the concepts they purport to measure [62]. In particular, they have a clear definition and can be easily collected. In this sense, the construct validity of the independent variable should be acceptable in our study.

7.2 Internal Validity

The first threat is the selection bias of the existing CPDP studies. To mitigate this threat, we conducted a search of the literature published between 2002 and 2017 from three sources: Google Scholar, the existing systematic literature reviews, and literature reading. Consequently, more than 70 CPDP studies were identified. This ensures that most, if not all, of the existing CPDP studies have been included in our study. Therefore, this threat has been minimized in our study.

The second threat is the unknown effect of the actual development process of the code. Our study used a binary variable indicating whether a module is defective to examine the usefulness of the existing CPDP studies. It is possible that some modules have no defects simply because they undergo more rigorous quality assurance procedures (e.g., inspections, walkthroughs, and testing). In other words, the actual development process may have a confounding effect on our findings. However, we believe that this confounding effect should not have a substantial influence, as similar findings are obtained from different target projects in our study. Nonetheless, this threat needs to be mitigated by controlled experiments in the future work.

7.3 External Validity

The first threat is that our findings may only apply to the CPDP models built with (structural) code metrics. In our experiment, we compared simple module size models against the existing CPDP models, most of which use (structural) code metrics to build the CPDP models. In other

words, (structural) code metrics are the dominant defect predictors used in the existing CPDP models. In this context, it is understandable that simple module size models perform well, as most code metrics are closely related with module size. In the past decade, however, many new metric sets such as process metrics [25, 70, 130], organizational structure metrics [75], and change burst metrics [76] have been shown to exhibit more predictive power the structural metrics. It is unclear whether simple module size models still perform well compared against the CPDP models built with those new metric sets. This problem needs to be investigated in the future work.

The second threat to the external validity of this study is that our findings may not be generalized to other projects. In our experiments, we use more than 50 projects, which are across a wide range of project types and scales, as the target projects. The experimental results drawn from these systems are quite consistent. Furthermore, the datasets collected from these projects are large enough to draw statistically meaningful conclusions. Nonetheless, since the subject target projects under study might not be representative of projects in general, we do not claim that our findings can be generalized to all projects. As reported in References [19], external validity in defect prediction is still an open problem. To mitigate this threat, there is a need to replicate our study across a wider variety of projects in the future.

8 CONCLUSIONS AND FUTURE WORK

The motivation of this study was to understand the benefits of the existing CPDP models w.r.t. simple module size models in defect prediction performance. To this end, we empirically compared their performance in defect prediction. To obtain a reliable conclusion, we used the same public datasets, the same performance indicators, and the prediction performance reported in the original CPDP studies to conduct the comparison. To our surprise, we found that most of the existing CPDP studies did not perform well compared against ManualDown and ManualUp, two simple module size models. This result informs that the progress in cross-project defect prediction is not being achieved as it might have been envisaged. Therefore, we strongly recommend that future CPDP studies should use simple module size models as the baseline models when developing new CPDP models. Otherwise, unintentionally misleading conclusions on advancing the state-of-the-art may be drawn, which may cause missed opportunities for progress in cross-project defect prediction. Note that our conclusions are based on two specific size models, ManualDown and ManualUp, whose application requires the availability of a complete target project.

Our study does not investigate whether simple module size models are more useful in helping developers find real bugs in a more easy way when compared with the existing CPDP models. As highlighted in References [53], to this end, there is a need to put them into the real world and let them be used by developers who work on a live code base. In the future work, we plan to examine which models are significant and practically important. In addition, our study does not distinguish defect types, as they are unavailable in the existing defect datasets. However, in practice, it is very helpful for practitioners to know on which defect types an existing CPDP model works well. This enables to determine the "predictable" defect types. Automatically predicting a common defect type would help to save a lot of maintenance resources, while predicting critical defect types would help to save a lot of production loss [68]. Therefore, in the future work, another interesting direction is to investigate on which defect types the existing CPDP models work well when compared with simple module size models.

ACKNOWLEDGMENTS

The authors thank the editor and anonymous reviewers for their very insightful comments and very helpful suggestions in greatly improving the quality of this article.

REFERENCES

[1] Aarti, Geeta Sikka, and Renu Dhir. 2017. An investigation on the effect of cross project data for prediction accuracy. *Int. J. Syst. Assur. Eng. Manage.* 8, 2 (2017), 352–377.

[2] Fumio Akiyama. 1971. An example of software system debugging. In *Proceedings of International Federation of Information Processing Societies Congress (IFIPS'71)*. 353–359.

[3] Harald Altinger, Steffen Herbold, Jens Grabowski, and Franz Wotawa. 2015. Novel insights on cross project fault prediction applied to automotive software. In *Proceedings of the 27th IFIP WG 6.1 International Conference on Testing Software and Systems (ICTSS'15)*. 141–157.

[4] Tiago L. Alves, Christiaan Ypma, and Joost Visser. 2010. Deriving metric thresholds from benchmark data. In *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM'10)*. 1–10.

[5] Sousuke Amasaki, Kazuya Kawata, and Tomoyuki Yokogawa. 2015. Improving cross-project defect prediction methods with data simplification. In *Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'15)*. 96–103.

[6] Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. Syst. Softw.* 83, 1 (2010), 2–17.

[7] Djuradj Babic. 2012. *Adaptive Software Fault Prediction Approach Using Object-oriented Metrics*. Ph.D. thesis. Florida International University.

[8] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J. Roy. Stat. Soc. Ser. B* 57, 1 (1995), 289–300.

[9] Lionel C. Briand, Walcelio L. Melo, and Jurgen Wust. 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans. Softw. Eng.* 28, 7 (2002), 706–720.

[10] Ana Erika Camargo Cruz and Koichiro Ochimizu. 2009. Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09)*. 460–463.

[11] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2013. Multi-objective cross-project defect prediction. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation (ICST'13)*. 252–261.

[12] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2015. Defect prediction as a multiobjective optimization problem. *Softw. Test. Verif. Reliabil.* 25, 4 (2015), 426–459.

[13] Qimeng Cao, Qing Sun, Qinghua Cao, and Huobin Tan. 2015. Software defect prediction via transfer learning based neural network. In *Proceedings of the 1st International Conference on Reliability Systems Engineering (ICRSE'15)*. 1–10.

[14] Çağatay Çatal. 2016. The use of cross-company fault data for the software fault prediction problem. *Turk. J. Electr. Eng. Comput. Sci.* 24, 5 (2016), 3714–3723.

[15] Lin Chen, Bin Fang, Zhaowei Shang, and Yuanyan Tang. 2015. Negative samples reduction in cross-company software defects prediction. *Inf. Softw. Technol.* 62 (2015), 67–77.

[16] Xiang Chen, Yingquan Zhao, Qiuping Wang, and Zhidan Yuan. 2018. MULTI: Multi-objective effort-aware just-in-time software defect prediction. *Inf. Softw. Technol.* 93 (2018), 1–13.

[17] Ming Cheng, Guoqing Wu, Hongyan Wan, Guoan You, Mengting Yuan, and Min Jiang. 2016. Exploiting correlation subspace to predict heterogeneous cross-project defects. *Int. J. Softw. Eng. Knowl. Eng.* 26, 9&10 (2016), 1571–1580.

[18] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates.

[19] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2012. Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Emp. Softw. Eng.* 17, 4&5 (2012), 531–577.

[20] Kalhed El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. 2001. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Trans. Softw. Eng.* 27, 7 (2001), 630–650.

[21] Franz Faul, Edgar Erdfelder, Albert-Georg Lang, and Axel Buchner. 2007. G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behav. Res. Methods* 39, 2 (2007), 175–191.

[22] George A. Ferguson and Yoshio Takane. 2005. *Statistical Analysis in Psychology and Education*. McGraw–Hill Ryerson Limited.

[23] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans. Softw. Eng.* 38, 6 (2012), 1276–1304.

[24] James A. Hanley and Barbara J. McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[25] Ahmed E. Hassan. 2009. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. 78–88.

[26] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. 2015. An empirical study on software defect prediction with a simplified metric set. *Inf. Softw. Techno.* 59 (2015), 170–190.

[27] Peng He, Bing Li, and Yutao Ma. 2014. Towards cross-project defect prediction with imbalanced feature sets. *CoRR* abs/1411.4228 (2014). arxiv:1411.4228 https://arxiv.org/abs/1411.4228

[28] Peng He, Bing Li, Deguang Zhang, and Yutao Ma. 2014. Simplification of training data for cross-project defect prediction. *CoRR* (2014). arxiv:1405.0773 https://arxiv.org/abs/1405.0773

[29] Peng He, Yutao Ma, and Bing Li. 2016. TDSelector: A training data selection method for cross-project defect prediction. *CoRR* abs/1612.09065 (2016). http://arxiv.org/abs/1612.09065

[30] Qing He, Biwen Li, Beijun Shen, and Xia Yong. 2015. Cross-project software defect prediction using feature-based transfer learning. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware (Internetware'15)*. 74–82.

[31] Zhimin He, Fayola Peters, Tim Menzies, and Ye Yang. 2013. Learning from open-source projects: An empirical study on defect prediction. In *Proceedings of the 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13)*. 45–54.

[32] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. 2012. An investigation on the feasibility of cross-project defect prediction. *Automat. Softw. Eng.* 19, 2 (2012), 167–199.

[33] Steffen Herbold. 2013. Training data selection for cross-project defect prediction. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering (PROMISE'13)*. 6:1–6:10.

[34] Steffen Herbold. 2017. sherbold/replication-kit-tse-2017-benchmark: Release of the replication kit. (May 2017). DOI: http://dx.doi.org/10.5281/zenodo.581178

[35] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. A comparative study to benchmark cross-project defect prediction approaches (unpublished).

[36] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. Correction of "A comparative study to benchmark cross-project defect prediction approaches." *CoRR* abs/1707.09281 (2017). arxiv:1707.09281 http://arxiv.org/abs/1707.09281

[37] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. Global vs. local models for cross-project defect prediction: A replication study. *Emp. Softw. Eng.* 22, 4 (2017), 1866–1902.

[38] Kim Herzig, Sascha Just, Andreas Rau, and Andreas Zeller. 2013. Predicting defects using change genealogies. In *Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE'13)*. 118–127.

[39] Seyedrebvar Hosseini and Burak Turhan. 2016. Search based training data selection for cross project defect prediction. In *Proceedings of the the 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'16)*. 3:1–3:10.

[40] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2017. A systematic literature review and meta-analysis on cross project defect prediction (unpublished).

[41] Xiaoyuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE'15)*. 496–507.

[42] Xiaoyuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. 2017. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans. Softw. Eng.* 43, 4 (2017), 321–339.

[43] Marian Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE'10)*. 9:1–9:10.

[44] Yasutaka Kamei, Takafumi Fukushima, Shane Mcintosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E. Hassan. 2016. Studying just-in-time defect prediction using cross-project models. *Emp. Softw. Eng.* 21, 5 (2016), 2072–2106.

[45] Arvinder Kaur and Kamaldeep Kaur. 2016. Value and applicability of academic projects defect datasets in cross-project software defect prediction. In *Proceedings of the 2nd International Conference on Computational Intelligence and Networks (CINE'16)*. 154–159.

[46] Ishleen Kaur and Neha Kapoor. 2016. Token based approach for cross project prediction of fault prone modules. In *Proceedings of the International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT'16)*. 215–221.

[47] Kazuya Kawata, Sousuke Amasaki, and Tomoyuki Yokogawa. 2015. Improving relevancy filter methods for cross-project defect prediction. In *Proceedings of the 3rd International Conference on Applied Computing and Information Technology/Proceedings of the 2nd International Conference on Computational Science and Intelligence (ACIT-CSI'15)*. 2–7.

[48] Taghi M. Khoshgoftaar, Pierre Rebours, and Naeem Seliya. 2009. Software quality analysis by combining multiple projects and learners. *Softw. Qual. J.* 17, 1 (2009), 25–49.

[49] Akif Güneş Koru, Khaled El Emam, Dongsong Zhang, Hongfang Liu, and Divya Mathew. 2008. Theory of relative defect proneness. *Emp. Softw. Eng.* 13, 5 (2008), 473–498.

[50] Akif Güneş Koru, Hongfang Liu, Dongsong Zhang, and Khaled El Emam. 2010. Testing the theory of relative defect proneness for closed-source software. *Emp. Softw. Eng.* 15, 6 (2010), 577–598.
[51] Akif Güneş Koru, Dongsong Zhang, Khaled El Emam, and Hongfang Liu. 2009. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Trans. Softw. Eng.* 35, 2 (2009), 293–304.
[52] Rahul Krishna, Tim Menzies, and Wei Fu. 2016. Too much automation? The bellwether effect and its implications for transfer learning. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*. 122–131.
[53] Michele Lanza, Andrea Mocci, and Luca Ponzanelli. 2016. The tragedy of defect prediction, prince of empirical software engineering research. *IEEE Softw.* 33, 6 (2016), 102–105.
[54] Luigi Lavazza and Sandro Morasca. 2016. An empirical evaluation of distribution-based thresholds for internal software measures. In *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'16)*. 6:1–6:10.
[55] Luigi Lavazza and Sandro Morasca. 2016. Identifying thresholds for software faultiness via optimistic and pessimistic estimations. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'16)*. 28:1–28:10.
[56] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E. James Whitehead Jr. 2013. Does bug prediction support human developers? Findings from a google case study. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 372–381.
[57] Zhiqiang Li, XiaoYuan Jing, Xiaoke Zhu, and Hongyu Zhang. 2017. Heterogeneous defect prediction through multiple kernel learning and ensemble learning. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME'17)*. 91–102.
[58] Zhiqiang Li, Xiao Yuan Jing, Fei Wu, Xiaoke Zhu, Baowen Xu, and Shi Ying. 2017. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. (unpublished).
[59] Yi Liu, Taghi M. Khoshgoftaar, and Naeem Seliya. 2010. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Trans. Softw. Eng.* 36, 6 (2010), 852–864.
[60] Baojun Ma, Huaping Zhang, Guoqing Chen, Yanping Zhao, and Bart Baesens. 2014. Investigating associative classification for software fault prediction: An experimental perspective. *Int. J. Softw. Eng. Knowl. Eng.* 24, 1 (2014), 61–90.
[61] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. 2012. Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.* 54, 3 (2012), 248–256.
[62] Sue McGregor. 1998. Property-based software engineering measurement. *IEEE Trans. Softw. Eng.* 22, 1 (1998), 68–86.
[63] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. 2011. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 26th International Conference on Automated Software Engineering (ASE'11)*. 343–351.
[64] Tim Menzies, Rahul Krishna, and David Pryor. 2016. The PROMISE repository of empirical software engineering data. Retrieved from http://openscience.us/repo.
[65] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. 2010. Defect prediction from static code features: Current results, limitations, new approaches. *Automat. Softw. Eng.* 17, 4 (2010), 375–407.
[66] Osamu Mizuno and Yukinao Hirata. 2014. A cross-project evaluation of text-based fault-prone module prediction. In *Proceedings of the 6th International Workshop on Empirical Software Engineering in Practice (IWESEP'14)*. 43–48.
[67] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker, and Kenichi Matsumoto. 2013. Assessing the cost effectiveness of fault prediction in acceptance testing. *IEEE Trans. Softw. Eng.* 39, 10 (2013), 1345–1357.
[68] Martin Monperrus. 2014. A critical review of "automatic patch generation learned from human-written patches": Essay on the problem statement and the evaluation of automatic software repair. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*. 234–242.
[69] Sandro Morasca and Luigi Lavazza. 2016. Slope-based fault-proneness thresholds for software engineering measures. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE'16)*. 12:1–12:10.
[70] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*. 181–190.
[71] Sara Moshtari and Ashkan Sami. 2016. Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC'16)*. 1415–1421.

[72] Sara Moshtari, Ashkan Sami, and Mahdi Azimi. 2013. Using complexity metrics to improve software security. *Comput. Fraud Secur.* 2013, 5 (2013), 8–17.

[73] Nachiappan Nagappan, Thomas Ball, and Brendan Murphy. 2006. Using historical in-process and product metrics for early estimation of software failures. In *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE'06)*. 62–74.

[74] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. 2006. Mining metrics to predict component failures. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. 452–461.

[75] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*. 521–530.

[76] Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. 2010. Change bursts as defect predictors. In *Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE'10)*. 309–318.

[77] Jaechang Nam and Sunghun Kim. 2015. CLAMI. Retrieved from https://github.com/lifove/CLAMI.

[78] Jaechang Nam and Sunghun Kim. 2015. CLAMI: Defect prediction on unlabeled datasets. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*. 452–463.

[79] Jaechang Nam and Sunghun Kim. 2015. Heterogeneous defect prediction. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE'15)*. 508–519.

[80] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 382–391.

[81] Paloma Oliveira, Marco Tulio Valente, and Fernando Paim Lima. 2014. Extracting relative thresholds for source code metrics. In *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14)*. 254–263.

[82] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'Union fait la force. In *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14)*. 164–173.

[83] Fayola Peters and Tim Menzies. 2012. Privacy and utility for defect prediction: Experiments with MORPH. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. 189–199.

[84] Fayola Peters, Tim Menzies, Liang Gong, and Hongyu Zhang. 2013. Balancing privacy and utility in cross-company defect prediction. *IEEE Trans. Softw. Eng.* 39, 8 (2013), 1054–1068.

[85] Fayola Peters, Tim Menzies, and Lucas Layman. 2015. LACE2: Better privacy-preserving data sharing for cross project defect prediction. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE'15)*. 801–811.

[86] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*. 409–418.

[87] Faimison Porto and Adenilso Simao. 2016. Feature subset selection and instance filtering for cross-project defect prediction: Classification and ranking. *CLEI Electr. J.* 19, 3 (2016), 1–17.

[88] Rahul Premraj and Kim Herzig. 2011. Network versus code metrics to predict defects: A replication study. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM'11)*. 215–224.

[89] Fangyun Qin, Zheng Zheng, Chenggang Bai, Yu Qiao, Zhenyu Zhang, and Cheng Chen. 2015. Cross-project aging related bug prediction. In *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS'15)*. 43–48.

[90] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. 2012. Recalling the "imprecision" of cross-project defect prediction. In *Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE'12)*. 61:1–61:11.

[91] Jeanine Romano, Jeffrey D. Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys. In *Proceedings of Annual Meeting of the Florida Association of Institutional Research*. 1–33.

[92] Duksan Ryu and Jongmoon Baik. 2016. Effective multi-objective naïve Bayes learning for cross-project defect prediction. *Appl. Soft Comput.* 49 (2016), 1062–1077.

[93] Duksan Ryu, Okjoo Choi, and Jongmoon Baik. 2016. Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Emp. Softw. Eng.* 21, 1 (2016), 43–71.

[94] Duksan Ryu, Jong In Jang, and Jongmoon Baik. 2015. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *J. Comput. Sci. Technol.* 30, 5 (2015), 969–980.

[95] Duksan Ryu, Jong In Jang, and Jongmoon Baik. 2017. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw. Qual. J.* 25, 1 (2017), 235–272.
[96] Ricardo Satin, Igor Scaliante Wiese, and Reginaldo Ré. 2015. An exploratory study about the cross-project defect prediction: Impact of using different classification algorithms and a measure of performance in building predictive models. In *Proceedings of the Latin American Computing Conference (CLEI'15)*. 1–12.
[97] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data quality: Some comments on the NASA software defect datasets. *IEEE Trans. Softw. Eng.* 39, 9 (2013), 1208–1215.
[98] Gwowen Shieh, Show Li Jan, and Ronald H. Randles. 2007. Power and sample size determinations for the Wilcoxon signed-rank test. *J. Stat. Comput. Simul.* 77, 8 (2007), 717–724.
[99] Pradeep Singh and Shrish Verma. 2015. Cross project software fault prediction at design phase. *Int. J. Comput. Electr. Automat. Control Inf. Eng.* 9, 3 (2015), 800–805.
[100] Pradeep Singh, Shrish Verma, and O. P. Vyas. 2013. Cross company and within company fault prediction using object oriented metrics. *Int. J. Comput. Appl.* 74, 8 (2013), 5–11.
[101] Jeffrey Stuckman, James Walden, and Riccardo Scandariato. 2017. The effect of dimensionality reduction on software vulnerability prediction models. *IEEE Trans. Reliabil.* 66, 1 (2017), 17–37.
[102] Mark D. Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. 2015. Replicating and re-evaluating the theory of relative defect-proneness. *IEEE Trans. Softw. Eng.* 41, 2 (2015), 176–197.
[103] Mathupayas Thongmak and Pornsiri Muenchaisri. 2003. Predicting faulty classes using design metrics with discriminant analysis. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP'03)*. 621–627.
[104] Burak Turhan, Ayşe Bener, and Tim Menzies. 2010. Regularities in learning defect predictors. In *Proceedings of the 11th International Conference on Product-Focused Software Process Improvement (PROFES'10)*. 116–130.
[105] Burak Turhan, Gozde Kocak, and Ayşe Bener. 2009. Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Syst. Appl.* 36, 6 (2009), 9986–9990.
[106] Burak Turhan, Tim Menzies, Ayş Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Emp. Softw. Eng.* 14, 5 (2009), 540–578.
[107] Burak Turhan, Ayşe Tosun Mısırlı, and Ayşe Bener. 2013. Empirical evaluation of the effects of mixed project data on learning defect predictors. *Inf. Softw. Technol.* 55, 6 (2013), 1101–1118.
[108] Satoshi Uchigaki, Shinji Uchida, Koji Toda, and Akito Monden. 2012. An ensemble approach of simple regression models to cross-project fault prediction. In *Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'12)*. 476–481.
[109] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th IEEE/ACM International Conference on Software Engineering (ICSE'16)*. 297–308.
[110] Shinya Watanabe, Haruhiko Kaiya, and Kenji Kaijiri. 2008. Adapting a fault prediction model to allow inter languagereuse. In *Proceedings of the 4th International Workshop on Predictive Models in Software Engineering (PROMISE'08)*. 19–24.
[111] Elaine J. Weyuker, Thomas J. Ostrand, and Robert M. Bell. 2010. Comparing the effectiveness of several modeling methods for fault prediction. *Emp. Softw. Eng.* 15, 3 (2010), 277–295.
[112] Peter A. Whigham, Caitlin A. Owen, and Stephen G. Macdonell. 2015. A baseline model for software effort estimation. *ACM Trans. Softw. Eng. Methodol.* 24, 3 (2015), 20:1–20:11.
[113] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometr. Bull.* 1, 6 (1945), 80–83.
[114] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE'14)*. 38:1–38:10.
[115] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing Chi Cheung. 2011. ReLink: Recovering links between bugs and changes. In *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'11)*. 15–25.
[116] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. HYDRA: Massively compositional model for cross-project defect prediction. *IEEE Trans. Softw. Eng.* 42, 10 (2016), 977–998.
[117] Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E. Hassan, and Naoyasu Ubayashi. 2016. Thresholds for size and complexity metrics: A case study from the perspective of defect density. In *Proceedings of the International Conference on Software Quality, Reliability and Security (QRS'16)*. 191–201.
[118] Guoan You, Feng Wang, and Yutao Ma. 2016. An empirical study of ranking-oriented cross-project software defect prediction. *Int. J. Softw. Eng. Knowl. Eng.* 26, 9&10 (2016), 1511–1538.
[119] Liguo Yu and Alok Mishra. 2012. Experience in predicting fault-prone software modules using complexity metrics. *Qual. Technol. Quant. Manage.* 9, 4 (2012), 421–434.

[120] Qiao Yu, Shujuan Jiang, and Junyan Qian. 2016. Which is more important for cross-project defect prediction: Instance or feature? In *Proceedings of the 1st International Conference on Software Analysis, Testing and Evolution (SATE'16)*. 90–95.

[121] Xiao Yu, Jin Liu, Mandi Fu, Chuanxiang Ma, and Guoping Nie. 2016. A multi-source TrAdaBoost approach for cross-company defect prediction. In *Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering (SEKE'16)*. 237–242.

[122] Feng Zhang, Iman Keivanloo, and Ying Zou. 2017. Data transformation in cross-project defect prediction. *Emp. Softw. Eng.* 22, 6 (2017), 3186–3218.

[123] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. 2016. Towards building a universal defect prediction model with rank transformed predictors. *Emp. Softw. Eng.* 21, 5 (2016), 2107–2145.

[124] Feng Zhang, Quan Zheng, Ying Zou, and Ahmed E. Hassan. 2016. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. 309–320.

[125] Hongyu Zhang and Hee Beng Kuan Tan. 2007. An empirical study of class sizes for large Java systems. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC'07)*. 230–237.

[126] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. 2015. An empirical study of classifier combination for cross-project defect prediction. In *Proceedings of the 39th Annual International Computers, Software & Applications Conference (COMPSAC'15)*. 264–269.

[127] Yuming Zhou, Baowen Xu, and Hareton Leung. 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. *J. Syst. Softw.* 83, 4 (2010), 660–674.

[128] Yuming Zhou, Baowen Xu, Hareton Leung, and Lin Chen. 2014. An in-depth study of the potentially confounding effect of class size in fault prediction. *ACM Trans. Softw. Eng. Methodol.* 23, 1 (2014), 10:1–10:51.

[129] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'09)*. 91–100.

[130] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2007. Predicting defects for eclipse. In *Proceedings of the 3rd International Workshop on Predictive Models in Software Engineering (PROMISE'07)*. 9:1–9:7.

[131] Thomas Zimmermann and Andreas Zeller. 2005. When do changes induce fixes? In *Proceedings of the 2nd International Workshop on Mining Software Repositories (MSR'05)*. 1–5.

Received December 2016; revised December 2017; accepted January 2018